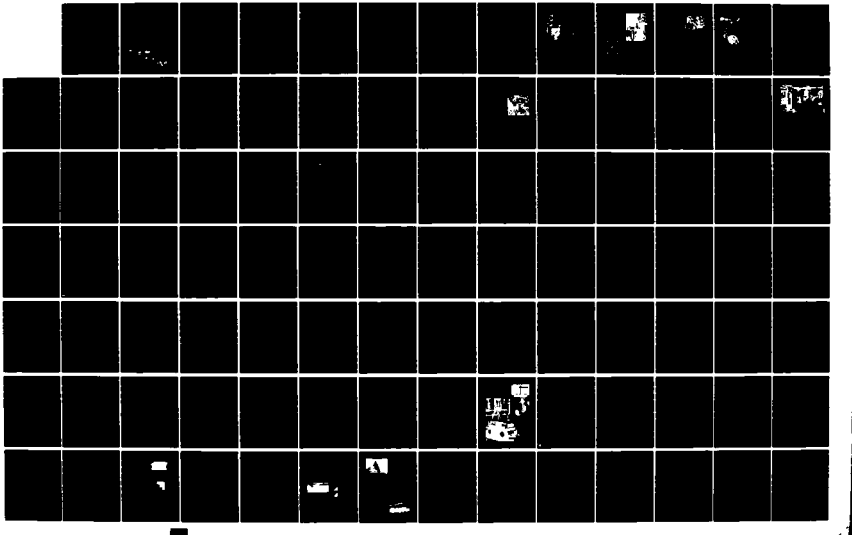
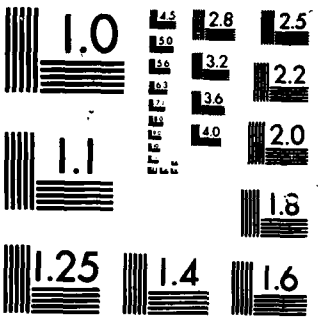
AD-A164 404 AUTONOMOUS MOBILE ROBOTS(U) CARNEGIE-MELLON UNIV 1/2
PITTSBURGH PA MOBILE ROBOT LAB H P MORAVEC 30 JAN 86
CMU-RI-MRL-86-1 N00014-81-K-0503

UNCLASSIFIED F/G 13/6 NL





(12)

Autonomous Mobile Robots

Annual Report - 1985

prepared by

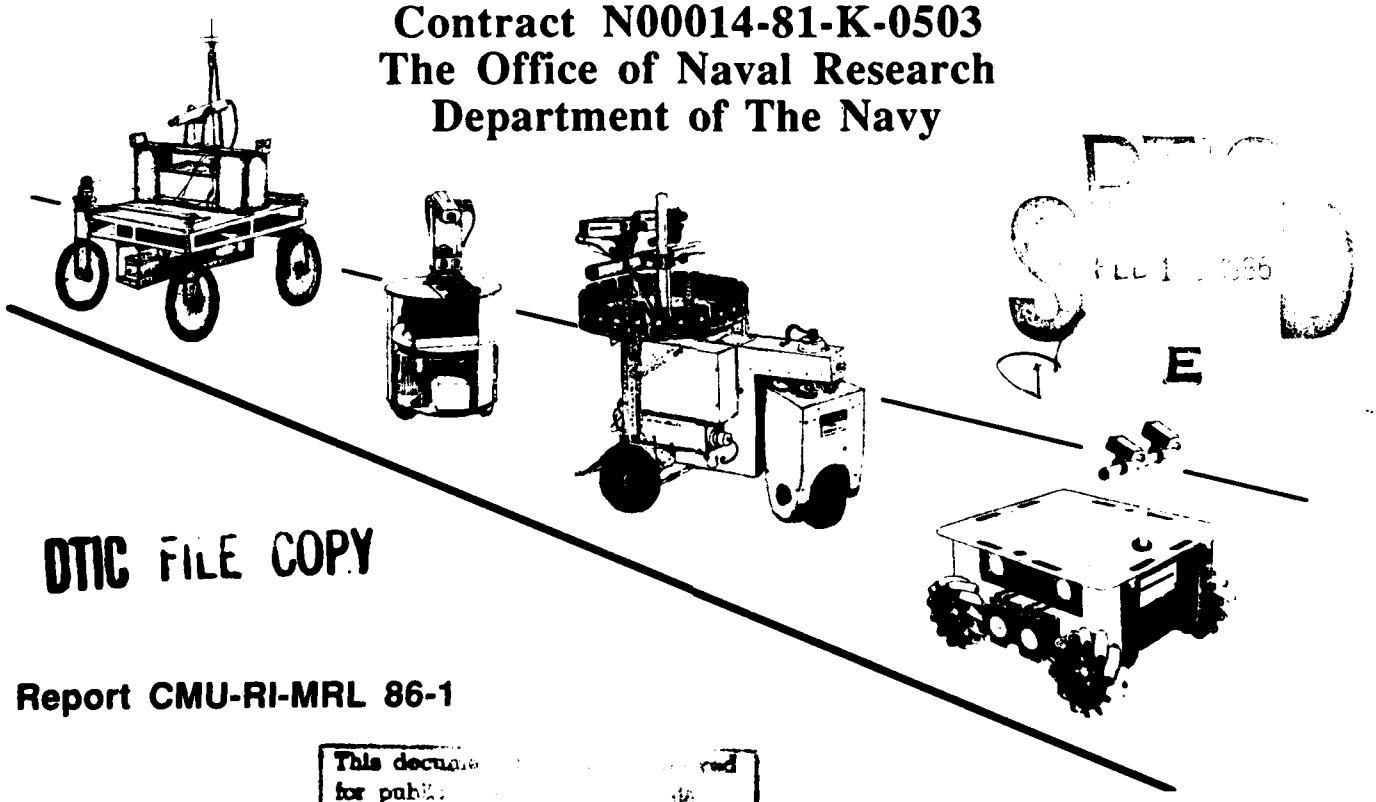
The Mobile Robot Laboratory

of

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

for

Contract N00014-81-K-0503
The Office of Naval Research
Department of The Navy



Report CMU-RI-MRL 86-1

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MRL-86-1	2. GOVT ACCESSION NO.: AD-A164404
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE (*and Subtitle*): Autonomous Mobile Robots	5. TYPE OF REPORT & PERIOD COVERED: Annual Report 1985
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR (*s*): Hans P. Moravec, ed.	8. CONTRACT OR GRANT NUMBER (*s*): N00014-81-K-0503
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University; Pittsburgh, PA 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit NR 610-001
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Dr. Alan R. Meyrowitz, Code 433, 800 N. Quincy St., Arlington, VA 22217	12. REPORT DATE: January 30, 1986
	13. NUMBER OF PAGES: 165
14. MONITORING AGENCY NAME & ADDRESS (*if different form Controlling Office*)	15. SECURITY CLASS. (*of this report*): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: None

16. DISTRIBUTION STATEMENT (*of this Report*)
Unlimited

17. DISTIBUTION STATEMENT (*of the abstract entered in Block 20; if different from Report*)
Unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (*Continue on reverse side if necessary and identify by block number*)
Mobile Robots, Vision, Stereo, Sonar, Navigation, Mapping, Path Planning, Uncertainty Handling, Road Following, Motion Control, Dynamic Control, Planning, Artificial Intelligence

20. ABSTRACT (*Continue on reverse side if necessary and identify by block number*)
Since 1981 the Mobile Robot laboratory of the Robotics Institute of Carnegie-Mellon University has conducted basic research in areas crucial for autonomous robots. We have built three mobile robots as testbeds for new concepts in control, vision, planning, locomotion and manipulation. This report recounts our work in 1985. Included are two papers describing two-dimensional sonar mapping and navigation, and a proposal for a three dimensional sonar. Three papers cover recent results in stereo visual navigation - we have achieved a tenfold speedup and a tenfold increase in navigational accuracy over our first generation system, and have a much deeper understanding of some of the mathematical foundations. (*continued on reverse side*)

(*abstract contd.*) Three papers describe results in a road navigation task - we are now able to navigate a simple road network at walking speeds with a single color camera on a roving robot using a variety of image processing and navigation methods. Three papers describe aspects of motion control, motors, wheeled kinematics and vehicle dynamics. Two papers present our newest robots, Neptune and Uranus. A final article gives some long term motivations and expectations for mobile robot research, and the report ends with a bibliography of our publications.

(*abstract contd.*) Three papers describe results in a road navigation task - we are now able to navigate a simple road network at walking speeds with a single color camera on a roving robot using a variety of image processing and navigation methods. Three papers describe aspects of motion control, motors, wheeled kinematics and vehicle dynamics. Two papers present our newest robots, Neptune and Uranus. A final article gives some long term motivations and expectations for mobile robot research, and the report ends with a bibliography of our publications.

The Mobile Robot Laboratory

Towards Autonomous Vehicles

The Mobile Robot Laboratory Staff

Introduction

The CMU Mobile Robot Lab was started in 1981 to pursue research in perception, planning and control for autonomously roving robots. The short and long range practical applications of robot mobility aside, we think our work directly addresses the problem of building a generally intelligent machine. Among living things, only mobile organisms exhibit the sensory and behavioral characteristics that are the basis of our own intelligence. A roving entity encounters a wide variety of circumstances, and must perceive and respond with great generality to function effectively. We feel our research makes discoveries that parallel the evolution of intelligence in mobile animals. The selection function in both cases is the same—the effective functioning of a physical mobile entity in a varied and uncertain world. We think this experimentally guided bottom up approach can find some solutions, such as the secret of effective common sense reasoning, more effectively than the seemingly direct traditional top down approach to artificial intelligence.

Our first funding came from an Office of Naval Research contract to develop land-based technology for eventual application to autonomous underwater robots. The subprojects were design and construction of highly maneuverable vehicles, development of stereo and sonar vision algorithms, and algorithms for path planning and higher level control. New developments were to be demonstrated in working systems that performed various tasks.

We chose two tasks, one simple and one complex. In the first, the vehicle was to travel to a goal location specified relative to its starting point, avoiding obstacles en route. This would encourage efforts in stereo, sonar, path planning, and vision-based vehicle motion estimation. The second task—finding, opening, and passing through doorways—was to serve as a longer term focus for work on maneuverable vehicles, object recognition, and distributed control.

Our first generation of obstacle avoidance systems now work, and we have taken first steps toward door-opening. We've built a simple vehicle to support obstacle avoidance work and a more complex vehicle to serve our longer term plans. Two obstacle avoidance systems have been tested, one relying solely on stereo and the other on sonar. An initial design for a distributed control system has been tested in simulation. We are preparing to start a second phase of our work which will extend the stereo capability towards shape extraction and merge stereo and sonar into a single system.

Overview

Our main subprojects pertain to vehicles, manipulators, servo control, stereo, sonar, and distributed processing. We will discuss each of these briefly before launching into the details.

Our long term plans call for an accurate, very maneuverable, self-powered vehicle carrying a small manipulator. *Pluto* (generically the *CMU Rover*) was designed to meet these requirements. Among its several innovations was an *omnidirectional* drive system for accurate control of robot motion in three independent degrees of freedom (forward/backward, left/right, and rotation). Our design used three complex wheel assemblies, each with two motors to independently drive and steer its own wheel. Coordinated control of the six motors was a more difficult problem than we had anticipated, and is now being attacked as a research problem in its own right.

For the sake of the vision and navigation research we constructed a much simpler second vehicle, *Neptune*. Power and control information come via a tether. Two synchronous AC motors steer and drive the robot, switched by a single onboard processor. Equipped with two vidicon cameras and a ring of sonar range finders, Neptune is robust and has been used in visual and sonar mapping, navigation and obstacle avoidance experiments.

There are several other hardware efforts in progress. We are building a third vehicle, *Uranus*, with a new, more easily controlled omnidirectional drive system to carry on the longer range work stalled in Pluto. We are working on a special-purpose manipulator for grasping doorknobs and have nearly completed a video digitizer/display that shares memory with a VAX. In addition, we are exploring processor and digitizer configurations for use on board the vehicles.

Pluto has been the center of our work on servo control. To control the motion of Pluto, we successfully designed and implemented an independent motor controller for each of its six motors. However, when we attempted to run the controllers simultaneously to obtain coordinated motion, the robot experienced severe oscillations because of dynamic coupling torques in the overconstrained wheelbase. These coupling effects could not be practically compensated using independent controllers executing on independent processors. The undesirable performance inspired us to work on the more general problem of the modeling and control of wheeled mobile robots. We are beginning the investigation by developing precise kinematic and dynamic models to be used as a basis for an integrated control strategy for Pluto's entire wheelbase. We plan to apply our modeling methodology to simulate wheeled mobile robots on a computer. This will enable us to test control strategies on

the computer simulated robot without the need for time-consuming hardware construction.

On the software side, we have concentrated on obstacle avoidance and distributed processing. We have two obstacle avoidance systems, one using stereo and the other using sonar. Both use a new path planner first developed for the stereo system. We have also designed and simulated the operation of a communication mechanism for distributed processors.

The stereo work improves on the system built for the Stanford Cart [7], which digitized nine images at each robot location and used correlation to track isolated feature points as the robot moved. We have reduced the number of images digitized per location, added constraints that improve the feature tracking ability, and are now modifying the motion estimation algorithm. In the process, we have reduced the runtime of the system by an order of magnitude. The robot can now visually navigate across a large room in under an hour on a VAX-11/780.

The sonar system uses data from a ring of twenty-four wide angle Polaroid range finders to map the surroundings of an autonomous mobile robot. A sonar range reading provides information concerning space occupancy in a cone subtending 30 degrees in front of the sensor. The reading is modelled as probability profiles projected onto a rasterized map of occupied and empty areas. Range measurements from multiple points of view (taken from multiple sensors on the robot, and from the same sensors after it moves) are systematically integrated in the map. Overlapping empty volumes reinforce each other, and empty volumes serve to condense the profiles of occupied volumes. The map resolution improves as more readings are added. The final map shows regions probably occupied, probably unoccupied, and unknown areas, with weights in each raster cell showing the confidence of these inferences. The method deals effectively with clutter, and can be used for motion planning and for extended landmark recognition.

The sonar and stereo systems both plan robot paths with a new algorithm called path relaxation. It was first developed for the stereo vision navigator, but coincidentally has a structure perfectly suited to our sonar mapper. Space is represented as a raster of weights as in the sonar maps. Costs are assigned to paths as a function of their length and the weights through which they pass. A combinatorial search on the raster grid coarsely finds a least cost path, then a relaxation procedure perturbs the coordinates of the vertices of this path to smooth it and reduce its cost.

Our work on distributed processing began with a design for a distributed planning and control system for the several processors of Pluto. A system has been designed around a network of message-passing kernels, a central blackboard module to represent state, and a notion of master/slave processes wherein masters monitor the blackboard while slaves handle external events. A small version of this system has been tested in simulation. We plan to give the design a more rigorous test soon with a distributed version of the sonar navigation system.

We have begun a new effort under the DARPA Autonomous Land Vehicles project in cooperation with other groups in the Robotics Institute led by William Whittaker and Takeo Kanade. The short term goal of this project is to build a system to follow roads; the long term goals include obstacle avoidance, off-road travel, object recognition, and long range navigation. The vehicle for this project is the Terregator, a large mobile robot built by Whittaker's group.

Vehicles

Our research plans called for a flexible vehicle to support work on vision, vision-guided manipulation, and the planning issues that come with mobility. Part of the design philosophy was the perception that a mobile wheelbase could be considered part of an attached arm. The weight and power of the arm can be reduced by using the mobility of the vehicle as an enormous reach substitute for the arm's shoulder joint. Such a strategy works best if the vehicle is given a full three degrees of freedom (forward/backward, left/right and compass heading) in the plane of the floor. Conventional steering arrangements as in cars give only two degrees at any instant. This approach to manipulation is most effective when the wheels can be servoed very accurately and rapidly.

Other properties we desired in a robot were that it run untethered, that it use multiple sensory systems, and that it carry some onboard processing power to reduce the communications bandwidth and perform some local decision-making.

Pluto, our first vehicle, was built to the above specifications. A second, simpler vehicle called Neptune was subsequently built to support obstacle avoidance work. A third vehicle, Uranus, is currently being designed to test a new concept in omnidirectionality.

Pluto

Physically, Pluto is cylindrical, about 1 meter tall, 55 centimeters in diameter, and weighs about 200 pounds (Figure 1a). Its three individually steerable wheel assemblies give it a full three degrees of mobility in the plane (Figure 1b). The control algorithm for this arrangement steers the wheels so that lines through their axles always meet at a common point. Properly orchestrated, this design permits unconstrained motion in any (2D) direction and simultaneous independent control of the robot's rotation about its own vertical axis.

To permit low-friction steering while the robot is stationary, each assembly has two parallel wheels connected by a differential gear (Figure 1c). The drive shaft of the differential goes straight up into the body of the robot, and a concentric hollow shaft surrounding the drive shaft connects to the housing of the differential. Turning the inner shaft causes the wheels to roll forward or backward; turning the outer one steers the assembly, causing the two wheels to roll in a circle.

Each shaft is driven by a brushless DC motor with samarium-cobalt permanent-magnet rotors and three-phase windings. The motor sequencing signals come directly from onboard microprocessors, one for each motor, which read shaft encoders to servo the motors to the desired motion. A seventh processor, the conductor, coordinates the action of the six motor sequencing processors. Another processor reads the shaft encoder outputs and monitors the motor torques to provide an onboard dead-reckoning estimate of the vehicle's position. Power for this ensemble is provided by a set of sealed lead-acid batteries.

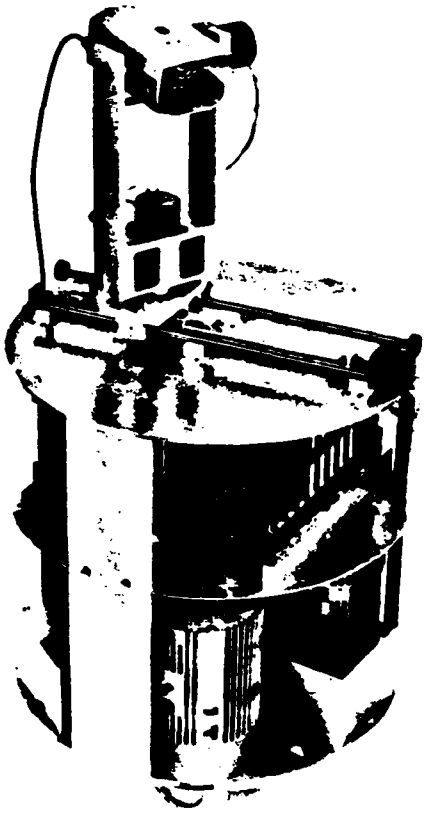


Figure 1a. Pluto

Pluto was to be equipped with a collection of sensors including cameras, sonar, and bump detectors and was to be used in a series of advanced experiments in vision, navigation and planning. The bulk of the computation would be performed on a remote VAX-11/780, with communication taking place over a microwave link for video and a radio link for other data. Extra processors were included in the design to service the sensors and manage the communication link.

This plan has been waylaid by a difficult and unexpected problem in controlling the six motors of the omnidirectional wheelbase. We are able to drive the robot successfully when one wheel at a time is energized, but large oscillations occur when all are running simultaneously. The problem is caused by interactions between the servo loops of the individual actuators through the redundant degrees of freedom in the wheels. A similar situation arises in a milder form in other locomotion systems with redundant degrees of freedom, especially legged vehicles. We are now investigating control algorithms and processor architectures for this problem, but in the meantime Pluto is unavailable for experimental work with our vision systems. Neptune was built to fill the gap.

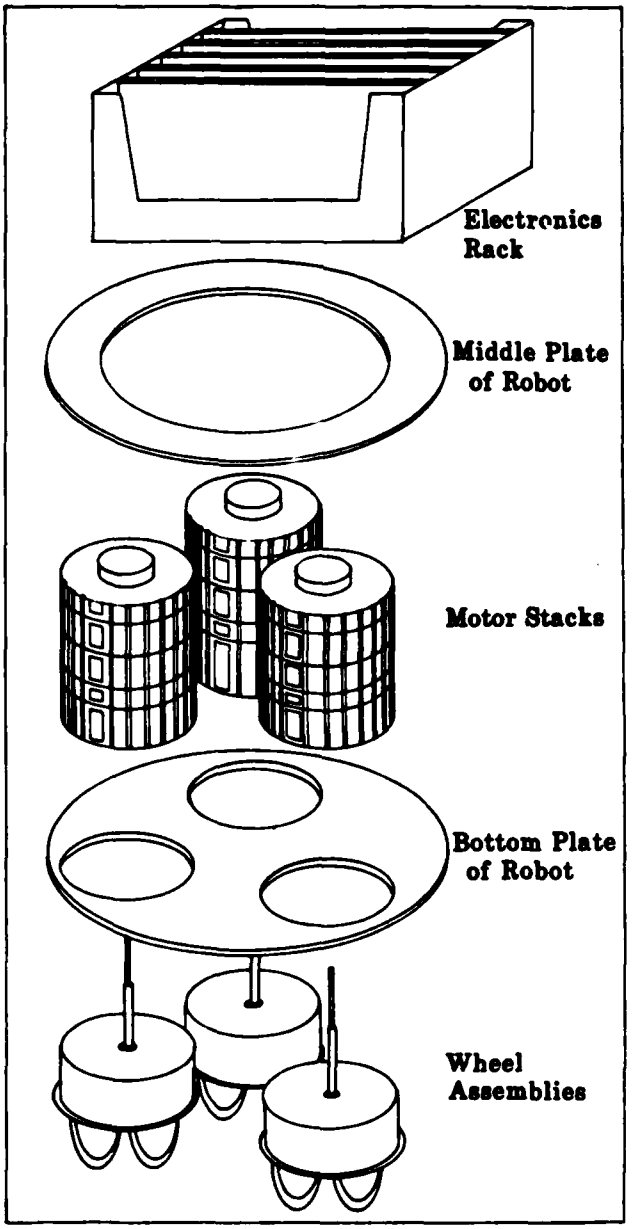


Figure 1b. Pluto subassembly: card cage, wheel assemblies, etc.

Neptune

We decided to build quickly, but carefully, a simple and robust platform for obstacle avoidance experiments. Neptune (Figure 2) was designed to eliminate many potential problems. It is a tethered, remotely powered tricycle with a lone onboard

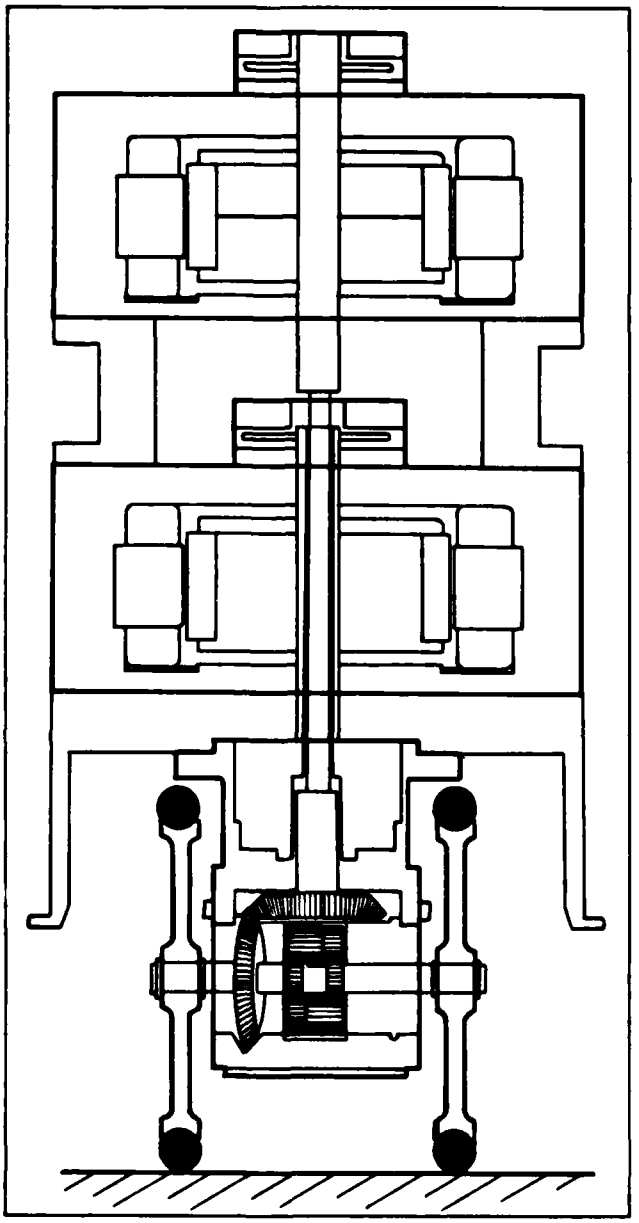


Figure 1c. Diagram of a wheel assembly illustrating differential gear, concentric drive shafts

processor. To simplify servoing and remove the need for shaft encoders, synchronous AC motors drive and steer the front wheel while the rear wheels trail. The vehicle is about 2 feet tall, 4 feet long, and 2 feet wide. It weighs about 250 pounds. It is currently configured with two black and white vidicon cameras on fixed mounts and a ring of twenty-four Polaroid sonar range-

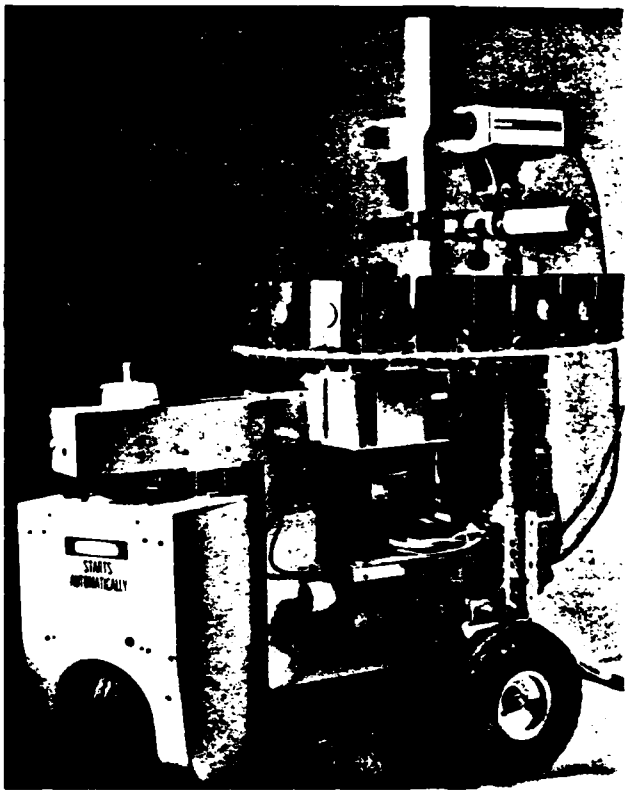


Figure 2. Neptune with sonar and stereo

finders. The range-finders have a useable range of about 35 feet and a 30 degree beam width, so that the beams of adjacent sensors overlap by about 50 percent. The vehicle moves at a constant velocity, with angles and distances controlled by timing the motors with an onboard MC68000.

Neptune is an unqualified success. It has been our workhorse for obstacle avoidance and indoor road following experiments and will be used in the future to test extended vision algorithms and to merge stereo and sonar into one system.

Uranus

Omnidirectionality appears to be an idea whose time has come. While Pluto was in gestation, several new methods for achieving omnidirectionality were published and patented. One, developed at Stanford, is based on novel wheels that have passive rollers instead of tires, oriented at right angles to the wheel (Figure 3a). The rollers permit the wheel to be pushed passively in the broadside direction. Three such wheels, each with its own motor, mounted around a round wheelbase allow smooth motion in three degrees of freedom. Regardless of the direction of travel, one wheel or another is always travelling nearly broadside, and this is a weakness of the system. It requires an expensive and potentially troublesome bearing system for the rollers, and suf-

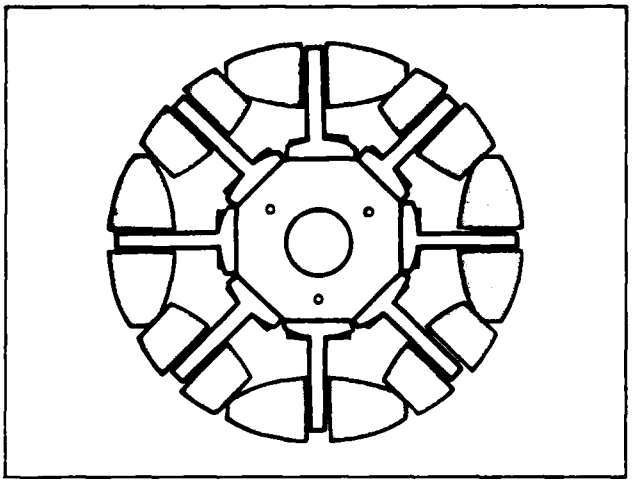


Figure 3a. Sketch of Stanford wheel

Figure 3b. Swedish designed wheels

fers from a low ground clearance limited by the roller diameter, and inability to travel on soft ground. Despite these limitations, it would have been a far more fortunate design choice than the individually steerable wheels of Pluto.

Another new design for omnidirectionality was invented recently in Sweden. It too uses wheels surrounded by passive rollers, but the rollers are angled at 45 degrees to the wheel plane (Figure 3b). One of these wheels can travel broadside on its rollers, but the whole wheel must simultaneously turn, resulting in a screw-like motion. Like screws, these wheels are not mirror symmetric and come in right handed and left handed varieties. An omnidirectional vehicle is built with four of these wheels, mounted like wagon wheels, but with careful attention to handedness. The right front wheel is right handed and the left front is left handed, but the right rear is left handed and the left rear is right handed (Figure 3c). Each wheel is turned by its own motor. To move the vehicle forward, all four wheels turn in the same direction, as in a conventional wagon. However, if the wheels on opposite sides of the vehicle are driven in opposite directions, the vehicle moves sideways, like a crab. By running the front and back wheels sideways in opposite directions, the vehicle can be made to turn in place. Because the rollers are not required to turn when the vehicle moves in the forward direction, the Swedish design has good bump and soft ground handling ability in that direction. In our experience-scarred judgement, the Swedish design is the most practical omnidirectional system. It is being used outside of an experimental context, in commercially available wheelchairs and factory transport vehicles.

Uranus, the Mobile Robot Lab's third construction, is being designed around this proven drive system to carry on the long range work stalled in Pluto. We obtained the wheels from Mecanum, Inc. of Sweden, which holds the license. Pluto's many lessons guide us in this project. In just about every way Uranus is simpler than Pluto. There are four motors, not six, no concentric shafts and only a single, benign, redundant interaction mode among the wheels.

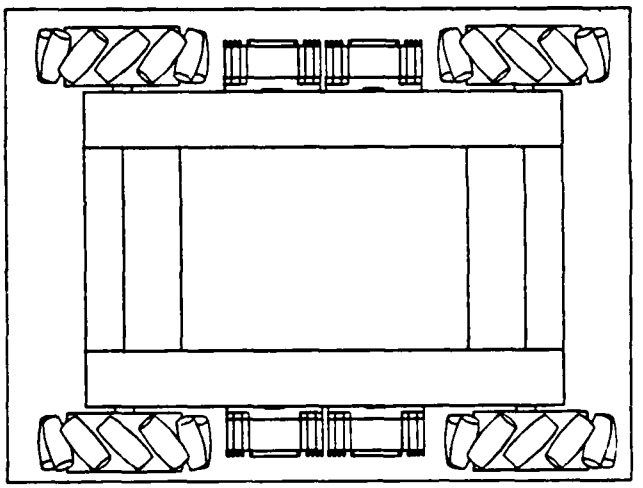


Figure 3c. Sketch illustrating handedness of wheels

A Manipulator for Door-opening

We have decided that visually locating, opening and passing through a door is an excellent task to guide development of advanced vision, planning and control work. To this end, we've designed and are building a special arm to be mounted on Uranus (Figure 4a).

The arm design is simultaneously strong, light and low-power because it exploits the mobility of the robot. The arm has four joints: a vertical translational joint, rotational shoulder and elbow joints with vertical axes, and a rotating wrist. The redundancies between the shoulder and elbow joints and the rotation of the vehicle permit the robot to hold the door in a stable, open position while the body of the robot passes through the doorway.

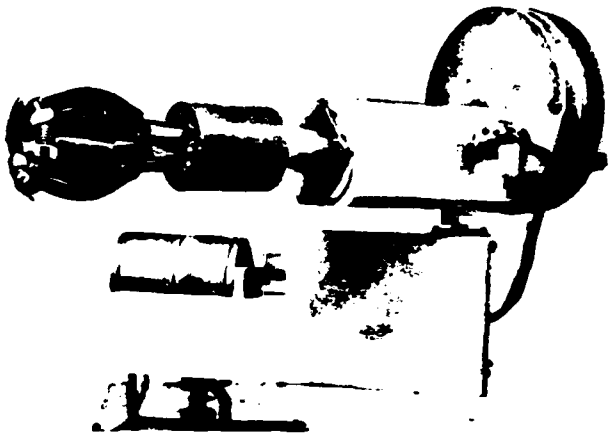


Figure 4a. Arm to be mounted on Uranus

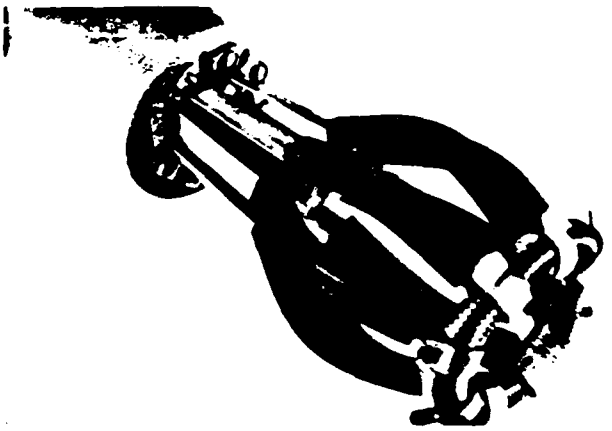


Figure 4b. Gripper and collar

The arm design uses the robot's strength to handle doors. The manipulator's joints are only lightly actuated, since the motors in the joints are used only for positioning the arm under no load. Once the gripper secures a doorknob, the elbow joint becomes a totally passive pivot and the base joint is alternately locked into position and released. Neither joint's motor is actuated again until the arm releases the door.

The gripper itself is constructed from a janitorial lightbulb extractor (Figure 4b). This is a spring-loaded, cylindrical device with a sliding collar. With the collar retracted, the gripper is pushed over the lightbulb (or doorknob); when the collar is tightened the gripper holds fast. Our manipulator uses this gripper with a motorized collar.

Mobility Control for Wheeled Mobile Robots

It has become clear to us that the complex mechanical designs of highly maneuverable wheeled mobile robots, such as Pluto and Uranus, require sophisticated coordinated controllers for effective motion control. Over-constrained multiple-wheeled robots, in particular, are a major challenge. We initially approached the problem by neglecting the motor interactions and designing independent control algorithms for each of the motors on Pluto. We found that only minimal mobility control was possible in this framework [9]. The severe motor interactions we observed provided a motivation to develop better control algorithms.

Pulse-Width Modulation Control of Brushless DC Motors

We implemented pulse-width modulation for controlling the brushless DC motors which actuate the wheels of Pluto [10]. The brushless DC motors utilize strong samarium-cobalt permanent magnets and are desirable for use on a mobile robot because of their high torque-to-weight ratio, ease of computer control, efficiency, and simple drive circuitry. We control each motor directly from a microprocessor using semiconductor power transistors. These devices operate very efficiently in the switching mode needed for pulse-width modulation.

Our theoretical and experimental results show that the motors can be modeled by linear discrete-time transfer functions, with the pulse-width playing the role of the control signal, if the pulse period is chosen much smaller than the time-constants of the motors. These models allow us to apply classical control engineering to the design of the motor control system. We have successfully designed controller structures and calculated feedback gains which provide each wheel with the ability to servo to a desired position and velocity within a specified time interval.

Wheeled Mobile Robot Simulations for Controller Design Studies

Our experience with Pluto prompted a systematic study of the problem of controlling wheeled mobile robots, both for Pluto's sake and for future designs. Our present approach to the problem is to develop precise kinematic and dynamic models of the robots. These models will form the basis of computer simulations of the robots on which proposed control strategies can be tested. Using computer simulations, we will have the ability to evaluate the performance of a robot/controller combination before spending much effort and expense in hardware construction. Adaptive control algorithms show promise for providing better robot control because they are able to adapt to coupling torques from other motors and to a changing floor surface or robot load. The controllers which demonstrate the best performance in simulations will be implemented on actual robots to verify both the accuracy of the simulations and the performance of the controllers.

Stereo Vision

The obstacle avoidance task prompted our first major work on robot perception. At the broadest level, the perception problem has two main components: understanding how to use individual sensors and understanding how to combine multiple sensors in a single system. We have addressed the first problem by developing rudimentary navigation systems that use vision and sonar separately. These systems are described in this and the following

section. Our work on integrating these two systems is only just beginning and will not be described in this paper.

Our stereo system continues the work done by Moravec with the Stanford Cart [7]. The basic task requires the robot to navigate from its initial position to a specified goal location, avoiding any obstacles in between. Stereo is used to detect obstacles and estimate the motion of the vehicle (actually avoiding the obstacles is discussed later under path planning). The Cart approach is to detect local, high variance *features* in one image, to use stereo correspondence to determine the three-dimensional positions of the features, and to track the features over time to determine the motion of the vehicle. Our work with these algorithms has focussed on the following issues:

- the number of stereo images used at each point in time
- the *interest operator* used to pick features
- the algorithm used for tracking

After reviewing the algorithms used by the Stanford Cart, we will discuss each of these issues in turn.

Vision in the Stanford Cart

The Stanford Cart used nine-way stereo at each robot position to detect and track obstacles. These images were obtained by stopping the robot and translating a single camera in two inch steps along a slider mechanism. An *interest operator* was applied to the center image to pick features, then a coarse to fine correlation process was applied to locate the features in the other eight images. Histogram-based triangulation from the set of match locations provided the initial three-dimensional obstacle positions. Obstacles were tracked as the robot moved by applying the correlator to the new center image to reacquire the old features. Then the features were matched in the other eight new images to obtain distances to the obstacles from the new robot location. Finally, least squares was used to find a best fit transformation mapping the old feature locations into the new, thereby obtaining the vehicle motion. Figure 5a illustrates the process of picking, matching, and tracking features through two steps of vehicle motion. The whole system moved the Cart in one-meter steps, taking about 15 minutes per step on a DEC KL-10.

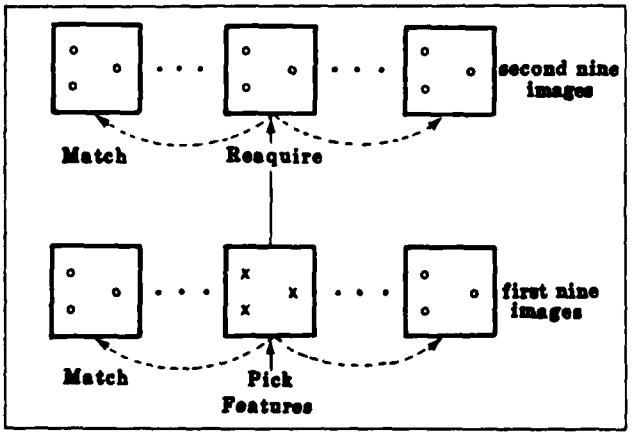


Figure 5a. Stanford Cart stereo matching

Number of Images

The great expense of using nine images prompted the use of only two-camera stereo in our current system. Since the redundancy provided by the nine images was a major strength of the original system, this decision initially lowered the reliability of the matching algorithm; to compensate, the stereo matcher now makes fuller use of constraints which reduce the search area in the second image. The constraints are as follows (Figure 5b). Between a stereo pair, the known camera geometry restricts possible matches to lie on a single line in the second image (the "epipolar line"). This line is the intersection of the image plane

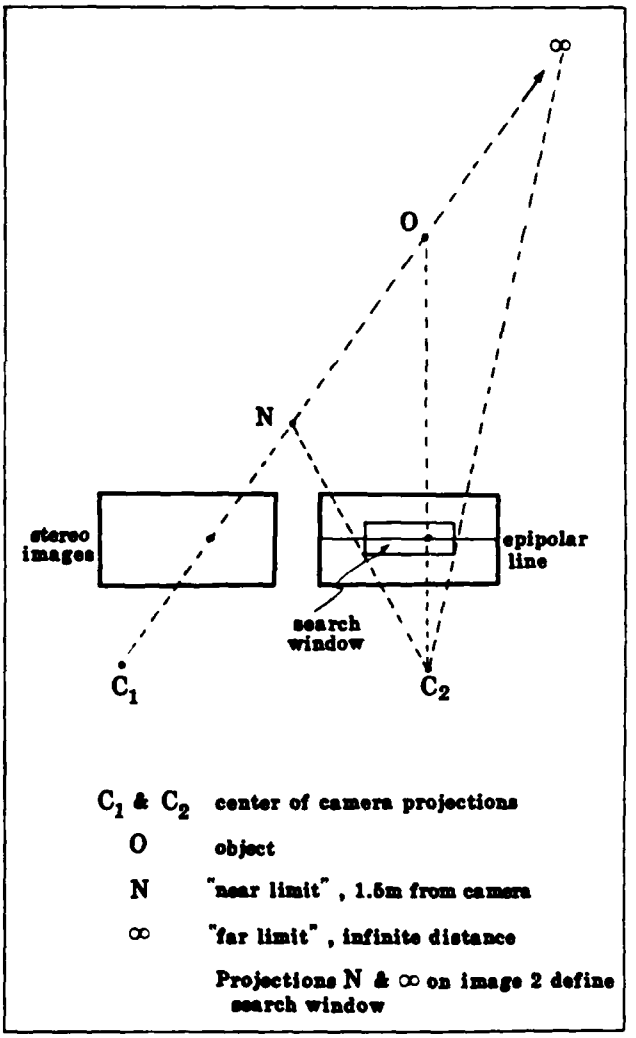


Figure 5b. Diagram of the epipolar and disparity constraints

of the second camera with the plane containing the obstacle and the two centers of projection. Near and far limits on the distance to an obstacle of 1.5 meters to infinity impose "disparity limits" that further restrict the search to a segment of the epipolar line. None of these constraints *per se* are available when features are reacquired in a new set of images. However, the known position of the obstacles together with an estimate of the vehicle motion still permit searches to be restricted to subwindows of the new images.

We have found that when all of the constraints are used, the qualitative system performance, measured in terms of the percentage of features matched correctly and the accuracy of motion estimates, is as good with the two-camera system as it was with the old system of nine images. The new system runs in about 35 CPU seconds per step (three to four minutes of elapsed time) on a VAX-11/780.

Although this experience demonstrates the effectiveness of two-camera stereo, the use of redundant images remains an interesting question. Two particular areas to be explored are the use of three cameras, which offers the ability to detect mismatches, and the use of the redundancy provided by motion. We expect to examine these areas in the future, both theoretically and empirically.

Interest Operators

The interest operator is designed to pick small patches or features in one image that can be reliably matched in another. In general, this requires that the patch exhibit high intensity variations in more than one direction to improve its localizability in another image. For example, edges show high variation in the direction of their gradient, but little variation in the direction of their length, making them poor to localize in that direction.

Ostensibly, a better interest operator will lead to a higher likelihood of correct matches. Many operators have been reported in the literature [11, 4], but no convincing evidence shows that any one operator is superior. Therefore, we evaluated the relative performance of a number of operators in the context of our system [16]. The operators used were those of Moravec [7], Kitchen and Rosenfeld [4], and several new operators we developed within our lab. As a control, a set of features were also picked by hand. The criterion used in assessing the performance of an operator was the number of features, from an initial set of forty picked by the operator, that could be correctly matched in another image. Here correct means that the match location was within a pixel or two of the best match subjectively as judged by the experimenter. Results were averaged over a number of trials with different images. Experiments were also run with and without the constraint offered by epipolar lines and disparity limits.

We found that rates of matching success showed very little variation between the better operators, which included the Moravec and Kitchen and Rosenfeld operators, and two of our new ones. The rates varied from about 60% correct in difficult images with no matching constraint, to over 90% when all constraints were used in less difficult images. On the whole, the Moravec operator performed slightly better than other operators and only a little worse than manual feature selection. More importantly, we found that the improvement bought by the use of search constraint was much more pronounced than that obtained by using different operators. We conclude that our research emphasis should no longer be placed on operators (since the Moravec operator is cheaper than, and at least as effective as other candidates), but should be placed on getting the most out of the available constraints and image redundancy.

Tracking and Motion Estimation

The Stanford Cart tracked features and estimated the motion of the vehicle as separate operations. Tracking was performed by searching for features one at a time in new images. Bad matches were then pruned with a heuristic that required the three-dimensional distances between pairs of features to remain the same over time. That is, objects that appeared to drift relative to other objects were deemed incorrect and were ignored. Motion estimation was then done by finding the transformation that minimized the least squared error between new and old feature positions.

This approach is unsatisfactory for two reasons. First, it makes poor use of the assumption that objects in the environment do not move. This is a valuable assumption and it underlies a large part of the Cart software; for example, it shows up in the pruning heuristic just mentioned and in the fitting of a single transformation to all feature points. The problem is that the constraint this assumption offers is employed only after feature match positions have been decided, which is too late. The correlator matches one feature at a time, without considering the locations of features matched previously; however, each new match decision implies constraint on possible locations for subsequent matches. Thus, the Cart algorithms allowed inconsistent matches to be made initially, then tried to catch them later. It would be preferable to ensure from the outset that matches were mutually consistent.

The second objection to the Cart approach is that it throws away image intensity information too early. Despite the best efforts of the interest operator, correlation peaks for individual features may be fairly broad, so that it makes little difference locally which pixel in a small region is chosen as the match. The actual location of the peak may be strongly influenced by noise in such cases. However, the correlator will pick the best peak and pass it on; a poor choice at this stage has the potential to skew both the depth estimate for the feature and the vehicle motion solution. It would be better to somehow capture the uncertainty in the match location and reflect that in other calculations.

We have addressed the first objection by using dead-reckoned estimates of vehicle motion to constrain the searches made by the matcher. This requires some tolerance to allow for errors in the dead-reckoned estimate, however, and in Neptune the tolerance must be fairly large. A better approach that addresses both objections has been developed by Lucas [5]. This is an iterative registration method that directly incorporates the assumption of stationary objects. An error measure for a trial transformation is defined to be the squared difference of image intensity between a feature in the previous image and its projected location in the new image, summed over all features. Starting from a dead-reckoned motion estimate, the known three-dimensional

feature positions are projected into the new image, the error measure is computed, and Newton iteration is employed to modify the transformation to minimize the error measure. Greater tolerance for errors in the initial estimate is obtained by applying the algorithm first to blurred versions of the image, then to successively sharper images. Lucas has shown that the algorithm works well, with synthetic and real images, for a single step of motion when the feature distances are given *a priori*. We are currently adapting the algorithm for use in our system.

We should note that another answer to our second objection is given by the work of Gennery [3], who used a correlator that estimated a two by two covariance matrix for the match location of a feature; that is, the matrix captured that broadness of the correlation peak. These matrices were propagated into covariance estimates for three-dimensional feature positions and for camera motion. We have not determined what role this idea will play in our future systems.

Sonar Mapping

Primarily because of computational expense, practical real-world stereo vision navigation systems [7, 14] build very sparse depth maps of their surroundings. Even with this economy, our fastest system [6] takes 30 to 60 seconds per one meter step on a 1 mips (millions of instructions per second) machine. Direct sonar range measurements promised to provide basic navigation and denser maps with considerably less computation. The readily available Polaroid ultrasonic range transducer [13] was selected, and a ring of 24 of these sensors was mounted on Neptune. We find sonar sensors interesting also because we would like to investigate how qualitatively different sensors, such as a sonar array and a pair of cameras, could cooperate in building up a more complex and rich description of the robot's environment.

Approach

Multiple wide-angle sonar range measurements are combined to map the surroundings of an autonomous mobile robot. A sonar range reading provides information concerning empty and occupied volumes in a cone subtending 30 degrees in front of the sensor. The reading is modelled as probability profiles (Figure 6a) projected onto a rasterized map, where occupied and empty areas are represented. Range measurements from multiple points of view (taken from multiple sensors on the robot, and from the same sensors after robot moves) are systematically integrated in the map. As more readings are added, the area deduced to be empty expands, and the expanding empty area encroaches on and sharpens the possibly occupied region. The map becomes gradually more detailed. The final map shows regions probably occupied, probably unoccupied, and unknown areas. The method deals effectively with clutter and can be used for motion planning and for extended landmark recognition. It was tested in cluttered environments using Neptune.

For navigation and recognition we developed a way of convolving two sonar maps, giving the displacement and rotation that best brings one map into registration with the other, along with a measure of the goodness of the match. The sonar maps are very useful for motion planning. They are denser than those made

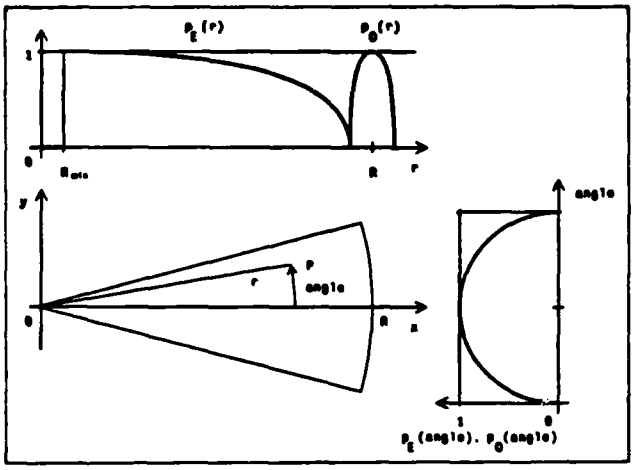


Figure 6a. Sonar beam probability profiles

by our stereo vision programs and computationally about an order of magnitude faster to produce. We are using them with the path relaxation method [15] to plan local paths for our robot.

The Sensor

The sonar devices being used are Polaroid laboratory grade ultrasonic transducers [13]. These sonar elements have a useful measuring range of one to thirty-five feet. The main lobe of the sensitivity function corresponds to a beam angle of 30° at -38 dB. Experiemental results showed that the range accuracy of the sensors is on the order of 1% . We are using the control circuitry provided with the unit, which is optimized for giving the range of the nearest sound reflector in its field of view and works for our purpose.

The Array

The sonar array, built at Denning Mobile Robotics and mounted on the Neptune, is composed of:

- a ring of 24 Polaroid sonar elements spaced 15° apart and mounted at a height of 31 inches above the ground (see Figure 2);
- a Z80 controlling microprocessor which selects and fires the sensors, times the returns, and provides a range value;
- a serial line over which range information is sent to a VAX mainframe that interprets the sonar data and performs the higher level mapping and navigation functions.

Representing the Sonar Beam

Because of the wide beam angle, individual rangings provide only indirect information about the location of the detected objects. We combine the constraints from individual readings to reduce the uncertainty. Our inferences are represented as probabilities in a discrete grid.

A range reading is interpreted as providing information about space volumes that are probably EMPTY and somewhere OCCUPIED. We model the sonar beam by probability distribution functions (Figure 6a). Informally, these functions model our confidence that the various points inside the cone of the beam are empty ($P_e(r)$), and our uncertainty about the location of the point, somewhere on the range surface of the cone, that caused the echo ($P_o(r)$). The functions are based on the range reading and on the spatial sensitivity pattern of the sonar and are a maximum near the center axis of the beam and taper to zero near the edges. These probability density functions are projected on a horizontal plane to generate map information. We use the profiles that correspond to a horizontal section of the sonar beam.

Building Maps

Sonar Maps are two-dimensional arrays of cells corresponding to a horizontal grid imposed on the area to be mapped. The final map has cell values in the range (-1,1), where values less than 0 represent probably empty regions, exactly zero represents unknown occupancy, and greater than 0 implies a probably occupied space (Figure 6b). This map is computed in a final step from two separate arrays analogous to the empty and occupied probability distributions introduced above. The position and the orientation of the sonar sensor at the time of the reading are used to register the profiles of each beam with the map. In Figure 6b, each symbol represents a square area six inches on a side. Empty areas with a high certainty factor are represented by white space; lower certainty factors by "+" symbols of increasing, thickness. Occupied areas are represented by "x" symbols, and unknown areas by ".". The robot positions where scans were taken are shown by circles, and the outline of the room and of major objects by solid lines.

Different readings asserting that a cell is EMPTY will enhance each other, as will readings implying that the cell may be OCCUPIED, while evidence that the cell is EMPTY will weaken the certainty of it being OCCUPIED and vice-versa. The operations performed on the empty and occupied probabilities are not symmetrical. The probability distribution for *empty* areas represents a solid volume whose totality is probably empty, but the *occupied* probability distribution for a single reading represents a lack of knowledge about the location of a single reflecting point somewhere in the range of the distribution. Empty regions are simply added using a probabilistic addition formula. The *occupied* probabilities for a single reading, on the other hand, are reduced in the areas that the other data suggests is empty, then normalized to make their sum unity. Only after this narrowing process are the *occupied* probabilities from each reading combined using the addition formula.

One range measurement contains only a small amount of information. By combining the evidence from many readings as the robot moves in its environment, the area known to be empty is expanded. The number of regions somewhere containing an occupied cell increases, while the range of uncertainty in each such region decreases. The overall effect, as more readings are added, is a gradually increasing coverage along with an increasing precision in object locations. Typically after a few hundred readings (and less than a second of computer time), our process

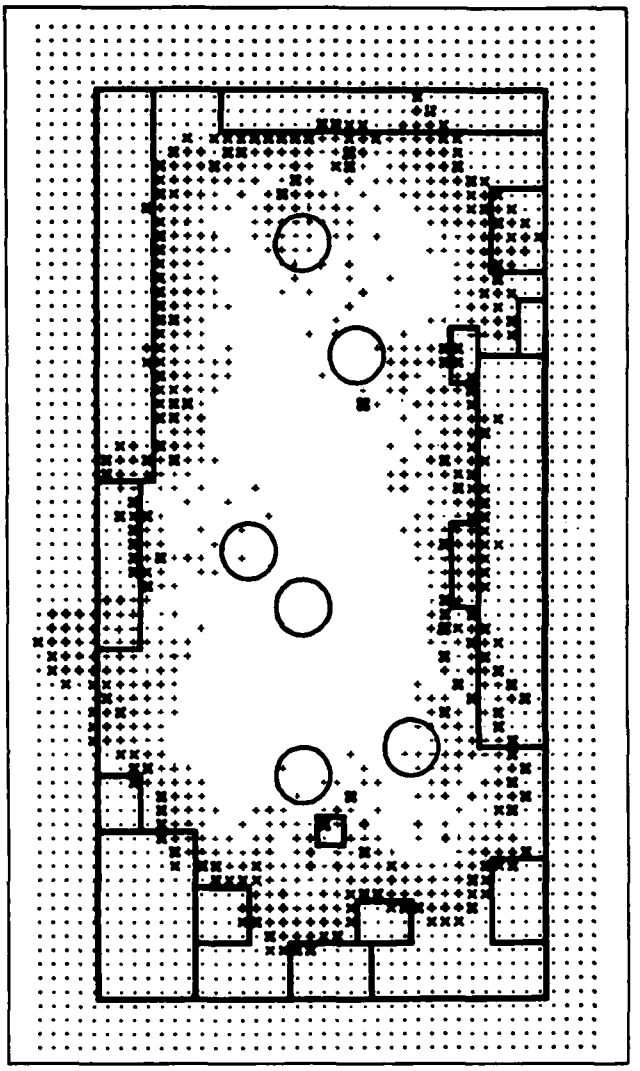


Figure 6b. A two-dimensional sonar map

is able to "condense out" a comprehensive map covering a thousand square feet with better than one foot position accuracy of the objects detected. Note that such a result does not violate information theoretic or degree of freedom constraints, since the detected boundaries of objects tend to be linear, not quadratic in the dimensions of the map. A thousand square foot map typically contains as little as a hundred linear feet of boundary.

Map Matching

We have also developed a procedure that can match two maps and report the displacement and rotation that best takes one into the other. We begin with the maps described above, with cell

values that are negative if the cell is empty, positive if occupied, and zero if unknown.

A measure of the goodness of the match between two maps at a trial displacement and rotation is found by computing the sum of products of corresponding cells in the two maps. An occupied cell falling on an occupied cell contributes a positive increment to the sum, as does an empty cell falling on an empty cell (the product of two negatives). An empty cell falling on an occupied one reduces the sum, and any comparison involving an unknown value causes neither an increase nor a decrease. This naive approach is very slow. Applied to maps with a linear dimension of n, each trail position requires $O(n^2)$ multiplications. Each search dimension (two axes of displacement and one of rotation) requires $O(n)$ trial positions. The total cost of the approach thus grows as $O(n^5)$. With a typical n of 50, this approach can use up a good fraction of an hour of VAX time.

Considerable savings come from the observation that most of the information in the maps is in the occupied cells alone. Typically only $O(n)$ cells in the map, corresponding to wall and object boundaries, are labelled occupied. A revised matching procedure compares maps A and B through trial transformation T (represented by a 2 x 2 rotation matrix and a 2 element displacement vector) by enumerating the occupied cells of A, transforming the coordinates of each such cell through T to find a corresponding cell in B. The [A, B] pairs obtained this way are multiplied and summed, as in the original procedure. The occupied cells in B are enumerated and multiplied with corresponding cells in A, found by transforming the B coordinates through T' (the inverse function of T), and these products are also added to the sum. The result is normalized by dividing by the total number of terms. This procedure is implemented efficiently by preprocessing each sonar map to give both a raster representation and a linear list of the coordinates of occupied cells. The cost grows as $O(n^4)$, and the typical VAX running time is down to a few minutes.

A further speedup is achieved by generating a hierarchy of reduced resolution versions of each map. A coarser map is produced from a finer one by converting two by two subarrays of cells in the original into single cells of the reduction. Our existing programs assign the maximum value found in the subarray as the value of the result cell, thus preserving occupied cells. If the original array has dimension n, the first reduction is of size $n/2$, the second of $n/4$, and so on. A list of occupied cell locations is produced for each reduction level so that the matching method of the previous paragraph can be applied. The maximum number of reduction levels is $\log_2 n$. A match found at one level can be refined at the next finer level by trying only about three values of each of the two translational and one rotational parameters, in the vicinity of the values found at the coarser level, for a total of 27 trials. With a moderate *a priori* constraint on the transformation this amount of search is adequate even at the first (coarsest) level. Since the cost of a trial evaluation is proportional to the dimension of the map, the coarse matches are inexpensive in any case. Applied to its fullest, this method brings the matching cost down to slightly larger than $O(n)$, and typical VAX times to under a second.

We found one further preprocessing step is required to make the matching process work in practice. Raw maps at standard resolutions (6 inch cells) produced from moderate numbers (about 100) of sonar measurements have narrow bands of cells labelled *occupied*. In separately generated maps of the same area, the relative positions of these narrow bands shift by as much as several pixels, making good registration of the occupied areas of the two maps impossible. This can be explained by saying that the high spatial frequency component of the position of the bands is noise; only the lower frequencies carry information. The problem is fixed by filtering (blurring) the occupied cells to remove the high frequency noise.

Experiments suggest that a map from 100 readings should be blurred with a spread of about two feet, while for maps made from 200 readings a one foot smear is adequate. Blurring increases the number of cells labelled *occupied*. So as not to increase the computational cost from this effect, only the final raster version of the map is blurred. The occupied cell list used in the matching process is still made from the unfiltered raster. With the process outlined here, maps with about 3000 six inch cells made from 200 well spaced readings can be matched with an accuracy of about six inches displacement and three degrees rotation in one second of VAX time.

Results

We incorporated the sonar map builder into a system that successfully navigates the Neptune robot through cluttered obstacle courses. The existing program incrementally builds a single sonar map by combining the readings from successive vehicle stops made about one meter apart. Navigation is by dead reckoning—we do not yet use the sonar map matching code. The next move is planned in the most recent version of the map by a path planning method based on path relaxation [15]. Since this method can cope with a probabilistic representation of occupied and empty areas and does path planning in a grid, it fits natually into our present framework. The system has successfully driven Neptune the length of our cluttered 30 by 15 foot laboratory using less than one minute of computer time.

Local Path Planning

Path relaxation is a two-step path planning process for mobile robots. It finds a safe path for a robot to traverse a field of obstacles and arrive at its destination. The first step of path relaxation finds a preliminary path on an 8-connected grid of points (Figure 7). The second step adjusts, or "relaxes," the position of each preliminary path point to improve the path.

One advantage of path relaxation is that it allows many different factors to be considered in choosing a path. Typical path planning algorithms evaluate the cost of alternative paths solely on the basis of path length. The cost function used by Path Relaxation, in contrast, also includes how close the path comes to objects (the further away, the lower the cost) and penalties for traveling through areas out of the field of view. The effect is to produce paths that neither clip the corners of obstacles nor make wide deviations around isolated objects, and that prefer to stay in mapped terrain unless a path through unmapped regions

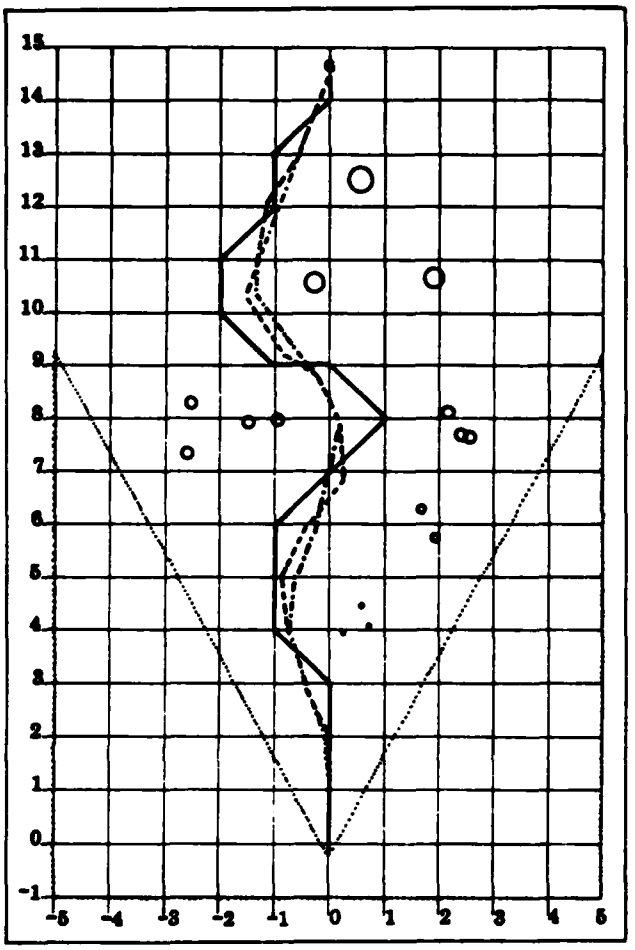


Figure 7. Path relaxation and 8-connectedness

is substantially shorter. Other factors, such as sharpness of corners or visibility of landmarks, could also be added for a particular robot or mission.

A cost function describes how desirable it is to have a path go through each point. This function can be thought of as a terrain map, with the vehicle as a marble rolling towards the goal. The terrain (cost function) consists of a gradual slope towards the goal, hills with sloping sides for obstacles, and plateaus for unexplored regions. The height of the hills has to do with the confidence that there really is an object there. Hill diameter depends on robot precision: A more precise robot can drive closer to an object, so the hills will be tall and narrow, while a less accurate vehicle will need more clearance, requiring wide, gradually tapering hillsides. Using this analogy, the first step of path relaxation is a global grid search that finds a good valley for the path to follow. The second step is a local relaxation step that moves the nodes in the path to the bottom of the valley in which they lie.

Grid Search

Once the grid size has been fixed, the next step is to assign costs to points on the grid and then to search for the best path along the grid from the start to the goal. "Best," in this case, has three conflicting requirements: shorter path length, greater margin away from obstacles, and less distance in uncharted areas. These three are explicitly balanced by the way path costs are calculated. A path's cost is the sum of the costs of the nodes through which it passes, each multiplied by the distance to the adjacent nodes. In a 4-connected graph all lengths are the same, but in an 8-connected graph we have to distinguish between orthogonal and diagonal links. The node costs consist of three parts to explicitly represent the three conflicting criteria.

- *Cost for distance.* Each node starts out with a cost of one unit, for length traveled.
- *Cost for near objects.* Each object near a node adds to that node's cost. The nearer the obstacle, the more cost it adds. The exact slope of the cost function will depend on the accuracy of the vehicle (a more accurate vehicle can afford to come closer to objects), and the vehicle's speed (a faster vehicle can afford to go farther out of its way), among other factors.
- *Cost for within or near an unmapped region.* The cost for traveling in an unmapped region will depend on the vehicle's mission. If this is primarily an exploration trip, for example, the cost might be relatively low. There is also a cost added for being near an unmapped region, using the same sort of function of distance as is used for obstacles. This provides a buffer to keep paths from coming too close to potentially unmapped hazards.

The first step of Path Relaxation is to set up the grid, construct the list of obstacles, and determine the vehicle's current position and field of view.[1] The system calculates the cost at each node, based on the distances to nearby obstacles and whether that node is within the field of view or not. The next step is to create a graph with links from each node to its 8 neighbors. The start and goal locations do not necessarily lie on grid points, so special nodes need to be created for them and linked into the graph.

The system then searches this graph for the minimum-cost path from the start to the goal. The search itself is a standard A* [12] search. The estimated total cost of a path, used by A* to pick which node to expand next, is the sum of the cost so far plus the straight-line distance from the current location to the goal. This has the effect, in regions of equal cost, of finding the path that most closely approximates the straight-line path to the goal.

Relaxation

Grid search finds an approximate path; the next step is an optimization step that fine-tunes the location of each node on the path to minimize the total cost. One way to do this would be to precisely define the cost of the path by a set of non-linear

[1] In this implementation, there are two types of obstacles: polygonal and circular. Currently, the circular obstacles are used for points found by stereo vision system, each bounded by a circular error limit, and the polygons are used for the field of view. The vision system will eventually give polygonal obstacles, at which point both the obstacles and the field of view will be represented as polygons and the circular obstacles will no longer be needed.

equations and solve them simultaneously to analytically determine the optimal position of each node. This approach is not, in general, computationally feasible. The approach used here is a relaxation method. Each node's position is adjusted in turn, using only local information to minimize the cost of the path sections on either side of that node. Since moving one node may affect the cost of its neighbors, the entire procedure is repeated until no node moves farther than some small amount.

Node motion has to be restricted. If nodes were allowed to move in any direction, they would all end up at low cost points, with many nodes bunched together and a few long links between them. This would not give a very good picture of the actual cost along the path. So in order to keep the nodes spread out, a node's motion is restricted to be perpendicular to a line between the preceding and following nodes. Furthermore, at any one step a node is allowed to move no more than one unit.

As a node moves, all three factors of cost are affected: distance traveled (from the preceding node, via this node, to the next node), proximity to objects, and relationship to unmapped regions. The combination of these factors makes it difficult to directly solve for minimum cost node position. Instead, a binary search is used to find that position to whatever accuracy is desired.

The relaxation step has the effect of turning jagged lines into straight ones where possible, of finding the "saddle" in the cost function between two objects, and of curving around isolated objects. It also does the "right thing" at region boundaries. The least cost path crossing a border between different cost regions will follow the same path as a ray of light refracting at a boundary between media with different transmission velocities. The relaxed path will approach that path.

Example Run

In Figure 8 we see a run using real data. Objects are represented as little circles, where the size of the circle is the positional uncertainty of the stereo system. The numbers are not all consecutive, because some of the points being tracked are on the floor or are high off the ground, and therefore are not obstacles. The dotted lines surround the area not in the field of view; this should extend to negative infinity. The start position of the robot is approximately (0, -.2) and the goal is (0, 14.5). The grid path is marked with 0's. After one iteration of relaxation, the path is marked by 1's. After the second relaxation, the path is marked by 2's. The greatest change from 1 to 2 was less than .3 meters, the threshold, so the process stopped. The size of the "hills" in the cost function is 1 meter, which means that the robot will try to stay 1 meter away from obstacles unless that causes it to go too far out of its way.

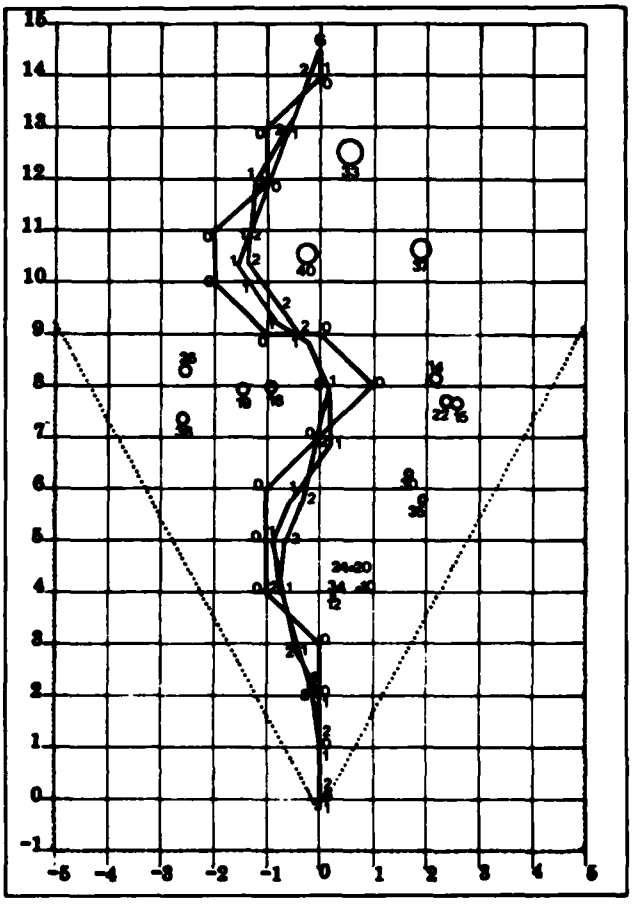


Figure 8. An example run

An Architecture for Distributed Control

Mobile robots pose a number of fascinating problems from the point of view of overall software system design. A large number of semi-independent activities are necessary to achieve autonomous mobility. These tasks include controlling actuators, monitoring several qualitatively different sensors, interpreting and integrating data from the sensors, and performing planning and problem-solving activities in several different areas and on various levels of abstraction.

These problems are aggravated by the fact that, to achieve real-time response, large amounts of processing power are necessary. One way of achieving this is to apply several processors to the problem. All this, however, brings the need to develop new and adequate distributed control and problem-solving mechanisms.

To face some of these concerns, we have designed a distributed software control structure [1] for mobile robots equipped with a variety of sensors and actuators. In this architecture, *Expert Modules* run as independent processes and exchange information over a *blackboard* (Figure 9a). The modules are distributed over a processor network and communicate through messages. We are now working on an experimental implementation of this system.

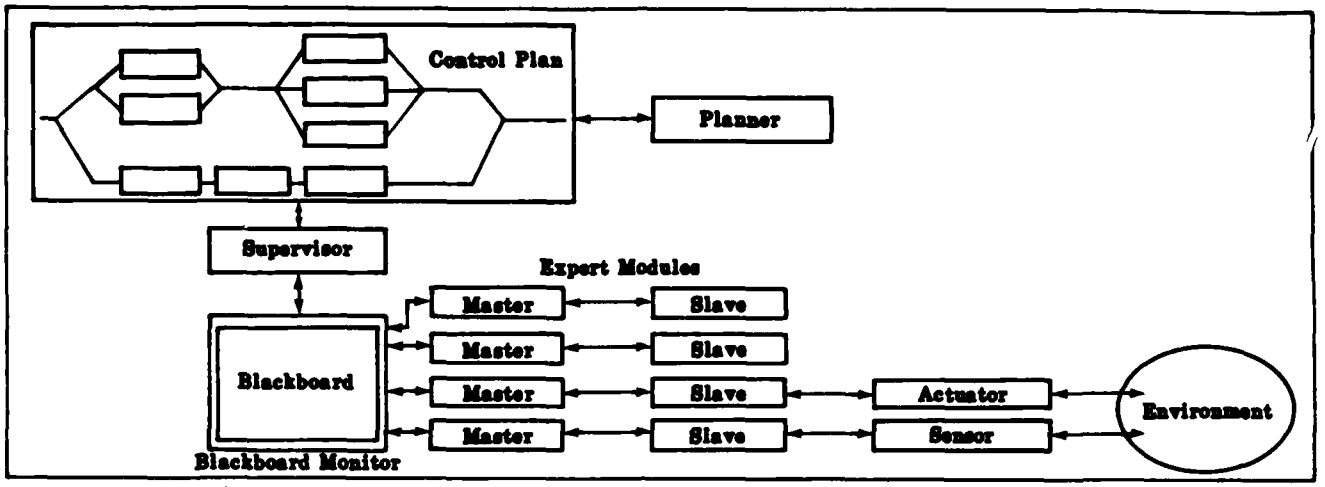


Figure 9a. General architecture of the distributed control system

The Architecture

Expert Modules are specialized subsystems used to control the operation of the sensors and actuators, interpret sensory and feedback data, build an internal model of the robot's environment, plan strategies to accomplish proposed tasks, and supervise the execution of the plan. Each Expert Module is composed of a *master* process and a *slave* process, where the master process controls the scheduling and the activities of the slave process and provides an interface to other modules. The master retrieves data from the blackboard that is needed by the slave, changes the status (*run/suspend/terminate/resume*) of the slave, and posts relevant results generated by the latter on the blackboard. The slave process is responsible for the processing and problem-solving activities as such.

One of the modules, the *supervisor*, dynamically abstracts scheduling information for the Expert Modules from a Control Plan. The Control Plan provides information specific to the execution of a given task by specifying subtasks and constraints in their execution. High-level information needed by the different subsystems is shared over the blackboard [2]. This includes information on the robot's status, relevant interpreted sensory and feedback data, and control information. Actual access to the blackboard is done only by the *blackboard monitor*, to insure the integrity of the posted data. A *blackboard scheduler* schedules the master processes to interact with the blackboard, according to their own priorities and the priorities of data and events being recorded there.

The Expert Modules are distributed over the processor network. An *executive* local to each processor is responsible for process scheduling. Besides using the blackboard, processes also exchange data of more specific interest directly among themselves. The system is built on top of a set of primitives that provide process handling, message-based interprocess communication and access to the blackboard.

An Example: Sonar-based Navigation

To provide an experimental testbed for the proposed architecture, we are re-implementing our sonar-based navigation system [8] as a distributed system. The main modules of the sonar system are sonar control, the scanner, the mapper, the path planner, and the conductor; for the distributed version we add to these a guardian and a supervisor process. The functions of these modules are:

Sonar Control: Interfaces to, and controls the sonar sensors. Provides range readings.

Scanner: Preprocesses the incoming sonar data and catches erroneous readings. Annotates sonar readings with sensor position, generating what is called a *view*.

Mapper: Integrates the view into a sonar map.

Path Planner: Using the information about free, unknown and occupied areas stored in a sonar map, generates safe paths for the robot.

Conductor: Performs the actual locomotion of the robot vehicle along the proposed path.

Guardian: Does a simple check on the sonar range data that is being acquired continuously during locomotion, to make sure that enough distance is maintained relative to objects in the robot's environment. This is a safety system to take care of rapidly moving objects that were not registered in the sonar map.

Supervisor: Takes care of the overall behavior of the system and extracts scheduling information from the Control Plan.

The original, monolithic version of the system worked by passing control to each module in sequence. However, such a serial-

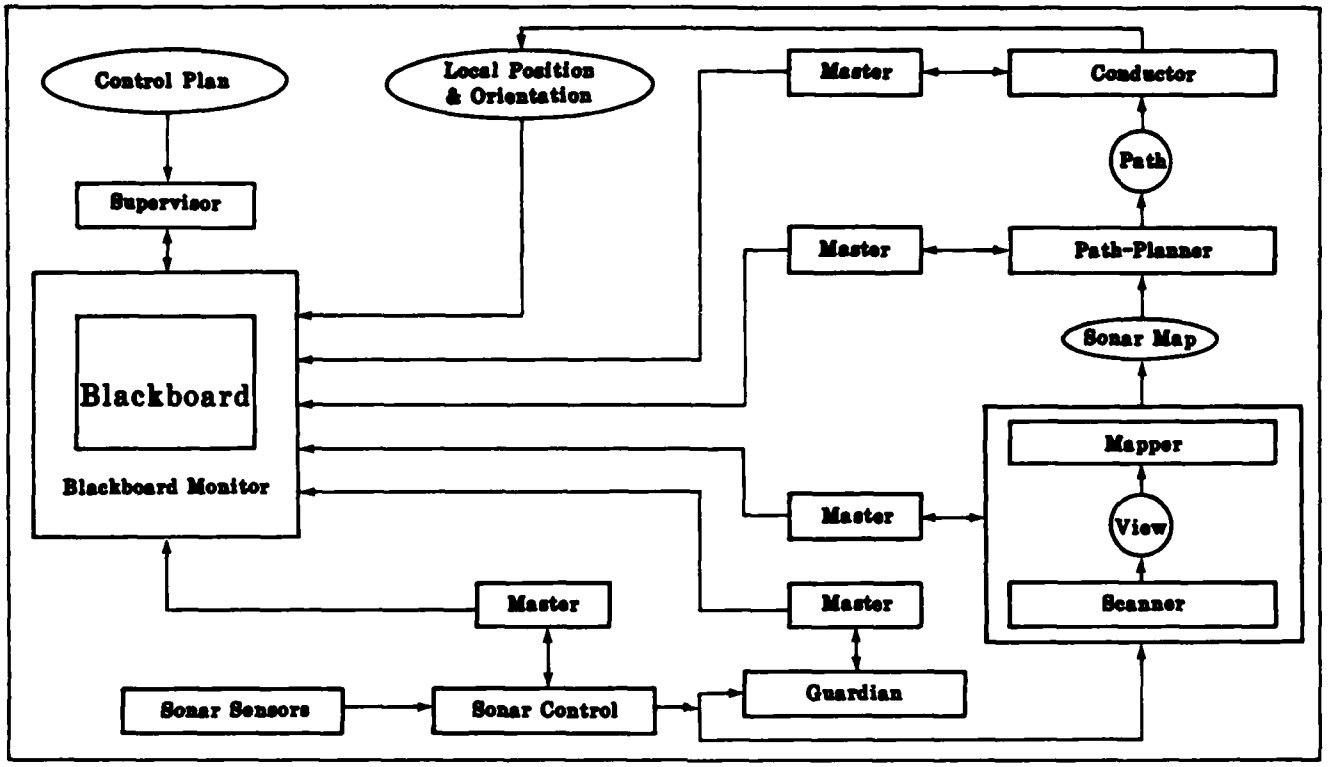


Figure 9b. A distributed implementation of the sonar-based mobile robot navigation system

ization is inconvenient when the processes involved are logically distinct or when they operate on different time-scales. For example, the path planner certainly requires the output of the mapper, but the planning activity is distinct from mapping and there is no reason why planning and mapping should follow a rigid pattern. They can be viewed instead as working on different sides of a shared database, with one process adding to and correcting the database while the other draws inferences from the information therein. As an example of different time-scales, both the guardian and the scanner act on sonar readings, but the guardian runs continuously whereas the scanner waits until its views come from sufficiently different positions of the robot.

In the distributed version of the system, each of the modules described above is an expert, with a master process that watches the blackboard for conditions that warrant a change in status (run/terminate/suspend/resume) of its slave. Information concerning the availability of data or results, the status of the robot, the activities of the Expert Modules and other relevant high-level data and control information is shared over the blackboard. The supervisor provides additional scheduling information to achieve an overall integrated and coherent behaviour. The bulk of the data is still passed directly between the modules themselves, since it consists of information relevant only to specific routines. Figure 9b illustrates the main flow of data control.

In a typical run, sonar ranging is done continuously. All readings are checked by the guardian to see whether any object is dangerously near. Selected sets of readings, taken from sufficiently distinct positions, are processed by the scanner and the mapper to provide an improved sonar map. Path-planning is done, and the existing path is updated. Locomotion proceeds; if the guardian issues a warning, the robot stops immediately and only proceeds after reassessing the situation of its environment. With this architecture, the system is able to respond in an asynchronous fashion to the various needs for data processing and problem-solving as they arise.

New Work

We have begun work in a major new area; road following systems for the DARPA Autonomous Land Vehicles program. The goals of the DARPA program begin with following well defined roads with no intersections or obstacles, then progress to navigation and obstacle avoidance in road networks and eventually to navigation in open terrain.

We are working on this in cooperation with other Robotics Institute groups led by William Whittaker and Takeo Kanade. The vehicle for this project is the Terregator, a large mobile robot built by Whittaker's group. Powered by an onboard gaso-

line fueled generator, it is designed for long outdoor journeys and is equipped with a television camera and microwave TV link. We have written a program that drives it along benign, well-defined roads in real time, visually tracking the left and right edges. We are extending this work to more difficult roads, longer journeys, and faster speeds, and plan to incorporate obstacle detection, landmark recognition and long range navigation. The effort complements our other projects and is a natural application of a number of the techniques we have been developing.

Conclusion and Philosophy

The most consistently interesting stories are those about journeys, and the most fascinating organisms are those that move from place to place. These observations are more than idiosyncrasies of human psychology, but illustrate a fundamental principle. The world at large has great diversity, and a traveller constantly encounters novel circumstances and is consequently challenged to respond in new ways. Organisms and mechanisms do not exist in isolation, but are systems with their environments, and those on the prowl in general have a richer environment than those rooted to one place. Mobility supplies danger along with excitement. Inappropriate actions or lack of well-timed appropriate ones can result in the demise of a free roamer, say over the edge of a cliff, far more easily than of a stationary entity for whom particular actions are more likely to have fixed effects. Challenge combines with opportunity in a strong selection pressure that drives an evolving species that happens to find itself in a mobile way of life in certain directions quite different from those of stationary organisms. The last billion years on the surface of the earth has been a grand experiment exploring these pressures. Besides the fortunate consequence of our own existence, some universals are apparent from the results to date and from the record. In particular, intelligence seems to follow from mobility.

The same pressures seem to be at work in the technological evolution of robots and it may be that mobile robots are the best route to solutions for some of the most vexing unsolved problems on the way to true artificial intelligence—problems such as how to program common sense reasoning and learning from sensory experience. This opportunity carries a price: programs to control mobile robots are more difficult to get right than most, and the robot is free to search the diverse world looking for just the combination that will foil the plans of its designers. There is still a long way to go.

We believe that our real-world experimental approach is teaching us important lessons. It is our experience that many perceptual and control problems succumb to simple techniques, but that only a very small fraction of the plausible simple methods work in practice. Determining *which* methods work often cannot be decided theoretically, but can be decided readily by realistic experiments.

Acknowledgments

Hans Moravec is director of the Mobile Robot Lab. Contributors to this article were Alberto Elfes, Karen Hensley, Larry Matthies, Hans Moravec, Pat Muir, Gregg Podnar, and Chuck Thorpe; Larry Matthies served as editor. Kevin Dowling and Mike Blackwell participated in the work described in the article.

Major funding for this work has been provided by the Office of Naval Research under contract number N00014-81-K-0503.

Dr. Hans Moravec (front, second from right), Director of the Mobile Robot Laboratory and Research Scientist of Robotics, with research staff.

References

[1] Elfes, A., and S.N. Talukdar. "A Distributed Control System for the CMU Rover." In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. IJCAI, August 1983.

[2] Erman, L.D., et al. "The HEARSAY-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty." *Computing Surveys* 12, no. 2, June 1980.

[3] Gennery, Donald B. *Modelling the Environment of an Exploring Vehicle by Means of Stereo Vision*. Ph.D. thesis, Stanford University, June 1980.

[4] Kitchen, L., and A. Rosenfeld. "Gray-level Corner Detection." *Pattern Recognition Letters* 1:95-102, December 1982.

[5] Lucas, B.D. *Generalized Image Registration by the Method of Differences: Algorithms, Applications, and Architectures*. Ph.D. thesis, Carnegie-Mellon University, February 1984.

[6] Matthies, L.H., and C.E. Thorpe. "Experience with Visual Robot Navigation." In *Proceedings of IEEE Oceans 84*. IEEE, Washington, D.C., August 1984.

[7] Moravec, Hans P. *Obstacle Avoidance and Navigation in the Real World by a Seeing Rover Robot*. Ph.D. thesis, Stanford University, September 1980. Published as *Robot Rover Visual Navigation* by UMI Research Press, Ann Arbor, Michigan, 1981.

[8] Moravec, H.P., and A. Elfes. "High-Resolution Maps from Wide-Angle Sonar." In *International Conference on Robotics and Automation*. IEEE, March 1985.

[9] Muir, P.F. *Digital Servo Controller Design for Brushless DC Motors*. Master's thesis, Carnegie-Mellon University, April 1984.

[10] Muir, P.F., and C.P. Neuman. "Pulse-Width Modulation Control of Brushless DC Motors for Robotic Applications." In *Proceedings of the 27th Midwest Symposium on Circuits and Systems*. Morgantown, West Virginia, June 1984.

[11] Nagel, Hans-Helmut. *Displacement Vectors Derived from Second Order Intensity Variations in Image Sequences*. Mitteilung Ifl-HH-M-97/82, Ifl, March 1982.

[12] Nilsson, Nils J. *Principles of Artificial Intelligence*. Tioga, Palo Alto, California, 1980.

[13] Polaroid Corporation. *Ultrasonic Range Finders*. Polaroid Corporation, 1982.

[14] Thorpe, C.E. "The CMU Rover and the FIDO Vision and Navigation System." In *1983 Symposium on Autonomous Underwater Robots*. University of New Hampshire, Marine Systems Engineering Lab, UNH, May 1983.

[15] ___. *Path Relaxation: Path Planning for a Mobile Robot*. Technical Report CMU-RI-TR-84-5, CMU Robotics Institute, April 1984.

[16] ___. "An Analysis of Interest Operators for FIDO." In *Proceedings of the IEEE Workshop on Computer Vision*. Annapolis, Maryland, April 1984.

Sonar Mapping, Imaging and Navigation

High Resolution Maps from Wide Angle Sonar

Hans P. Moravec **Alberto Elfes**

The Robotics Institute

Carnegie-Mellon University

Abstract

We describe the use of multiple wide-angle sonar range measurements to map the surroundings of an autonomous mobile robot. A sonar range reading provides information concerning empty *and* occupied *volumes in a cone (subtending 30 degrees in our case) in front of the sensor. The reading is modelled as probability profiles projected onto a rasterized map, where somewhere occupied and everywhere empty areas are represented. Range measurements from multiple points of view (taken from multiple sensors on the robot, and from the same sensors after robot moves) are systematically integrated in the map. Overlapping empty volumes re-inforce each other, and serve to condense the range of occupied volumes. The map definition improves as more readings are added. The final map shows regions probably occupied, probably unoccupied, and unknown areas. The method deals effectively with clutter, and can be used for motion planning and for extended landmark recognition. This system has been tested on the* Neptune *mobile robot at CMU.*

1. Introduction

This paper describes a sonar-based mapping system developed for mobile robot navigation. It was tested in cluttered environments using the *Neptune* mobile robot [8], developed at the Mobile Robot Laboratory of the Robotics Institute, CMU. The Neptune system has been used successfully in several areas of research, including stereo vision navigation [5, 10] and path planning [11]. Other research in the laboratory includes the investigation of adequate high-level robot control structures, the use of distributed and parallel processing methods to improve the real-time response of the system, navigation in outdoor environments and the design and construction of more advanced robots with higher mobility.

Primarily because of computational expense, practical real-world stereo vision navigation systems [7, 10] build very sparse depth maps of their surroundings. Even with this economy our fastest system, described in [5], takes 30 - 60 seconds per one meter step on a 1 mips machine.

This work has been supported in part by Denning Mobile Robotics, Inc., by the Western Pennsylvania Advanced Technology Center and by the Office of Naval Research under contract number N00014-81-K-0503. The second author is supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, Brazil, under Grant 200.986-80; in part by the Instituto Tecnológico de Aeronáutica - ITA, Brazil; and in part by The Robotics Institute, Carnegie-Mellon University.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

Direct sonar range measurements promised to provide basic navigation and denser maps with considerably less computation. The readily available Polaroid ultrasonic range transducer [9] was selected and a ring of 24 of these sensors was mounted on Neptune.

We find sonar sensors interesting also because we would like to investigate how qualitatively different sensors, such as a sonar array and a pair of cameras, could cooperate in building up a more complex and rich description of the robot's environment.

1.1. Goals

We expected sonar measurements to provide maps of the robot's environment with regions classified as empty, occupied or unknown, and matches of new maps with old ones for landmark classification and to obtain or correct global position and orientation information.

1.2. Approach

Our method starts with a number of range measurements obtained from sonar units whose position with respect to one another is known. Each measurement provides information about empty and possibly occupied volumes in the space subtended by the beam (a thirty degree cone for the present sensors). This occupancy information is projected onto a rasterized two-dimensional horizontal map. Sets of readings taken both from different sensors and from different positions of the robot are progressively incorporated into the sonar map. As more readings are added the area deduced to be empty expands, and the expanding empty area encroaches on and sharpens the possibly occupied region. The map becomes gradually more detailed.

For navigation and recognition we developed a way of convolving two sonar maps, giving the displacement and rotation that best brings one map into registration with the other, along with a measure of the goodness of the match.

The sonar maps happen to be very useful for motion planning. They are denser than those made by our stereo vision programs, and computationally about an order of magnitude faster to produce. We presently use them with the Path Relaxation method [11] to plan local paths for our robot.

1.3. Related Work

Sonar is a developed technology but few applications until recently involved detailed map building. Traditional marine applications, camera autofocus systems, and some simple robot navigation schemes [2, 6] rely on sparse proximity measurements to accomplish their narrow goals.

The most advanced sonar systems used in marine intelligence operations locate sound sources passively [1]. Ultrasound systems used in medicine are typically active and build maps for human perusal, but depend on accurate physical models of the environments that the sound traverses [4], and work with very small beam widths, about 1° - 3°. Narrow beam widths, formed by phased array techniques, are also used in advanced side looking mapping sonar system for submersibles. An independent CMU sonar mapping effort [3] also used a narrow beam, formed by a parabolic reflector, in its attempts to build a line-based description.

In contrast the sonar sensors that we choose to employ have a wide beam, with an effective angle of about 30°.

2. The Sonar System

2.1. The Sensor

The sonar devices being used are Polaroid laboratory grade ultrasonic transducers [9]. These sonar elements have a useful measuring range of 0.9 to 35.0 ft. The main lobe of the sensitivity function corresponds to a beam angle of 30° at −38 dB. Experimental results showed that the range accuracy of the sensors is on the order of 1 %. We are using the control circuitry provided with the unit, which is optimized for giving the range of the nearest sound reflector in its field of view, and works well for this purpose.

2.2. The Array

The sonar array, built at Denning Mobile Robotics, and mounted on the *Neptune* mobile robot is composed of:

- A ring of 24 Polaroid sonar elements, spaced 15° apart, and mounted at an height of 31 inches above the ground (see Fig. 1).
- A Z80 controlling microprocessor which selects and fires the sensors, times the returns and provides a range value.
- A serial line over which range information is sent to a VAX mainframe that presently interprets the sonar data and performs the higher level mapping and navigation functions.

3. Sonar Mapping

3.1. Obtaining Reliable Range Data from the Sonar Sensor

We begin our map building by preprocessing the incoming readings to remove chronic errors. The following steps are used:

- **Thresholding**: Range readings above a certain maximum R_u are discarded. We observe that sonar readings caused by specular reflections are often near the maximum range of the device (R_{max}). With R_u slightly below R_{max}, many of these readings are discarded. The system becomes slightly myopic, but the overall quality of the map improves. Very large open spaces are detected by analyzing the set of distance values obtained from the sonar, and in this case the filtering is not done. A similar heuristic is applied for small readings: values below the minimum sensor range R_{min} are usually glitches and are discarded.
- **Averaging**: Several independent readings from the same sensor at the same position are averaged. The sonar readings are subject to error not only from reflections but also from other causes such as fluctuations in the effective sensitivity of the transducer. As a result readings show a certain dispersion. Averaging narrows the spread.
- **Clustering**: A set of readings from one sensor at a given position sometimes shows a clustering of the data around two different mean values. This happens when different readings are being originated by objects at staggered distances. We apply a simple clustering analysis to the data, and extract a mean value for each cluster for use in subsequent processing.

3.2. Representing the Sonar Beam

Because of the wide beam angle the filtered data from the above methods provides only indirect information about the location of the detected objects. We combine the constraints from individual readings to reduce the uncertainty. Our inferences are represented as probabilities in a discrete grid.

A range reading is interpreted as providing information about space volumes that are probably EMPTY and somewhere OCCUPIED. We model the sonar beam by probability distribution functions. Informally, these functions model our confidence that the various points inside the cone of the beam are empty and our uncertainty about the location of the point, somewhere on the range surface of the cone, that caused the echo. The functions are based on the range reading and on the spatial sensitivity pattern of the sonar.

Consider a position P = (x,y,z) belonging to the volume swept by the sonar beam. Let:

R be the range measurement returned by the sonar sensor,
ε be the mean sonar deviation error,
ω be the beam aperture,
$S = (x_s, y_s, z_s)$ be the position of the sonar sensor,
δ be the distance from P to S,
θ be the angle between the main axis of the beam and SP.

We now identify two regions in the sonar beam:

- **Empty Region**: Points inside the sonar beam ($\delta < R-\varepsilon$ and $\theta \le \omega/2$), that have a probability $p_E = f_E(\delta,\theta)$ of being empty.
- **Somewhere Occupied Region**: Points on the sonar beam front ($\delta \in [R-\varepsilon, R+\varepsilon]$ and $\theta \le \omega/2$), that have a probability $p_O = f_O(\delta,\theta)$ of being occupied.

Fig. 2 shows the probability profiles for a sonar beam that returned a range reading R. The horizontal crossection of the beam is associated with two probability distributions corresponding to the empty and the occupied probabilities.

The empty probability density function for a point P inside the sonar beam is given by:

$$p_E(x,y,z) = p[\text{ position } (x,y,z) \text{ is empty }] = E_r(\delta) \cdot E_a(\theta) \qquad (1)$$

Figure 1: The *Neptune* mobile robot, with a pair of cameras and the sonar ring, in our laboratory. Sonar maps of this lab can be seen in Figures 3 through 8.

where

$$I_r(\delta) = 1 - ((\delta - R_{min})/(R - \epsilon - R_{min}))^2 \text{ for } \delta \in [R_{min}, R-\epsilon] \quad (2)$$

$I_r(\delta) = 0$ otherwise.

And

$$I_a(\theta) = 1 - (2\theta/\omega)^2 \text{ for } \theta \in [-\omega/2, \omega/2] \quad (3)$$

The occupied probability density function for a point P on the beam front is given by:

$$P_O(x, y, z) = P[\text{position } (x, y, z) \text{ is } \text{occupied}] = O_r(\delta)\, O_a(\theta) \quad (4)$$

where:

$$O_r(\delta) = 1 - ((\delta - R)/\epsilon)^2 \text{ for } \delta \in [R-\epsilon, R+\epsilon] \quad (5)$$

$O_r(\delta) = 0$ otherwise.

And:

$$O_a(\theta) = 1 - (2\theta/\omega)^2 \text{ for } \theta \in [-\omega/2, \omega/2] \quad (6)$$

These probability density functions are now projected on a horizontal plane to generate map information. We use the profiles that correspond to a horizontal section of the sonar beam ($z = z_c$).

3.3. Representing Maps

Sonar Maps are two-dimensional arrays of cells corresponding to a horizontal grid imposed on the area to be mapped. The grid has $M \times N$ cells, each of size $\Delta \times \Delta$. The final map has cell values in the range $(-1,1)$, where values less than 0 represent probably empty regions, exactly zero represents unknown occupancy, and greater than 0 implies a probably occupied space. This map is computed in a final step from two separate arrays analogous to the empty and occupied probability distributions introduced above.

A cell is considered UNKNOWN if no information concerning it is available. Cells can be EMPTY with a degree of certainty or confidence $Emp(XY)$ and OCCUPIED with a degree of certainty $Occ(XY)$ both values ranging from 0 to 1.

The *a priori* empty and occupied certainty values for a given grid cell (XY) and reading are determined by taking the minimum of the reading's P_E and maximum of P_O, respectively, over the cell through a horizontal slice through the beam center.

3.4 Composing Information from Several Readings

The map is built by projecting the beam probabilities onto the discrete cells of the sonar map and there combining it with information from other beams. The position and the orientation of the sonar sensor are used to register the beam with the map.

Different readings asserting that a cell is EMPTY will enhance each other, as will readings implying that the cell may be OCCUPIED while evidence that the cell is EMPTY will weaken the certainty of it being OCCUPIED and vice-versa.

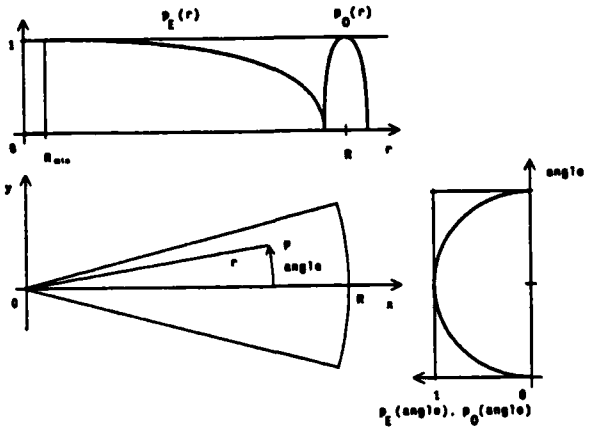


Figure 2: The Probability Profiles corresponding to the probably Empty and somewhere Occupied regions in the sonar beam. The profiles represent a horizontal cross section of the beam.

The operations performed on the empty and occupied probabilities are not symmetrical. The probability distribution for *empty* areas represents a solid volume whose totality is probably empty but the *occupied* probability distribution for a single reading represents a lack of knowledge we have about the location of a single reflecting point somewhere in the range of the distribution. Empty regions are simply added using a probabilistic addition formula. The *occupied* probabilities for a single reading, on the other hand, are reduced in the areas that the other data suggests is empty, then normalized to make their sum unity. Only after this narrowing process are the *occupied* probabilities from each reading combined using the addition formula.

One range measurement contains only a small amount of information. By combining the evidence from many readings as the robot moves in its environment, the area known to be empty is expanded. The number of regions somewhere containing an occupied cell increases, while the range of uncertainty in each such region decreases. The overall effect as more readings are added is a gradually increasing coverage along with an increasing precision in the object locations. Typically after a few hundred readings (and less than a second of computer time) our process is able to "condense out" a comprehensive map covering a thousand square feet with better than one foot position accuracy of the objects detected. Note that such a result does not violate information theoretic or degree of freedom constraints, since the detected boundaries of objects are linear, not quadratic in the dimensions of the map. A thousand square foot map may contain only a hundred linear feet of boundary.

Formally the evidence combination process proceeds along the following steps:

1. RESET: The whole Map is set to UNKNOWN by making $Emp(X,Y) := 0$ and $Occ(X,Y) := 0$.

2. SUPERPOSITION OF EMPTY AREAS: For every sonar reading k modify the emptyness information over its projection by:

   ENHANCE: $Emp(X,Y) :=$
   $Emp(X,Y) + Emp_k(X,Y) - Emp(X,Y) \times Emp_k(X,Y)$

3. SUPERPOSITION OF OCCUPIED AREAS: For each reading k, shift the occupied probabilites around in response to the combined emptyness map using:

   CANCEL: $Occ_k(X,Y) := Occ_k(X,Y) \cdot (1 - Emp(X,Y))$

   NORMALIZE: $Occ_k(X,Y) := Occ_k(X,Y) / \sum Occ_k(X,Y)$

   ENHANCE: $Occ(X,Y) :=$
   $Occ(X,Y) + Occ_k(X,Y) - Occ(X,Y) \times Occ_k(X,Y)$

4. THRESHOLDING: The final occupation value attributed to a cell is given by a thresholding method:

   THRESHOLD: $Map(X,Y) :=$
   $\begin{cases} Occ(X,Y) & \text{if } Occ(X,Y) \geq Emp(X,Y) \\ -Emp(X,Y) & \text{if } Occ(X,Y) < Emp(X,Y) \end{cases}$

3.5. Maps

A typical map obtained through this method is shown in Fig. 3, and the corresponding certainty factor distributions are shown in Figs. Fig. 4 and 5. These are the maps obtained before the thresholding step.

The final maps obtained after thresholding are shown in Figs. 6, 7 and 8.

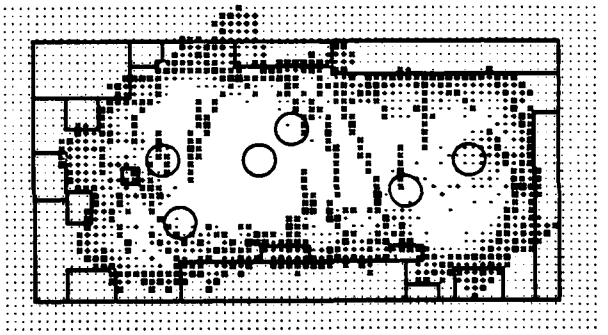


Figure 3: A Two-Dimensional Sonar Map. Each symbol represents a square area six inches on a side in the room pictured in Figure 1. The right edge of this diagram corresponds to the far wall in the picture. *Empty* areas with a high certainty factor are represented by white space; lower certainty factors by "+" symbols of increasing thickness. *Occupied* areas are represented by "×" symbols, and *Unknown* areas by "·". The robot positions where scans were taken are shown by circles and the outline of the room and of the major objects by solid lines.

4. Matching

Sonar navigation would benefit from a procedure that can match two maps and report the displacement and rotation that best takes one into the other.

Our most successful programs begin with the thresholded maps described above, with cell values that are negative if the cell is empty, positive if occupied and zero if unknown.

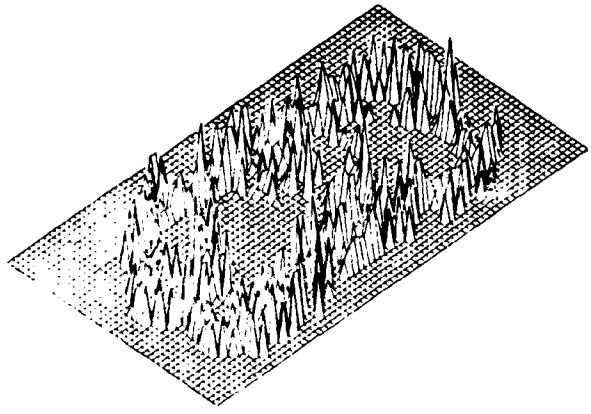


Figure 4: The Occupied Areas in the Sonar Map. This 3-D view shows the Certainty Factors $Occ(X,Y)$.

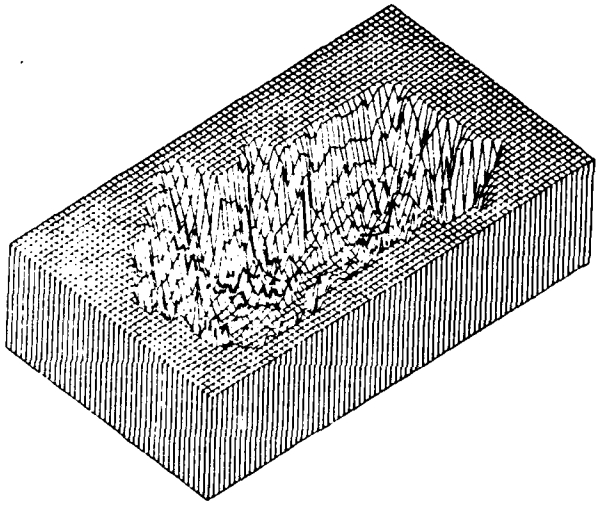


Figure 5: The Empty Areas in the Sonar Map. This 3-D view shows the Certainty Factors $Emp(X,Y)$.

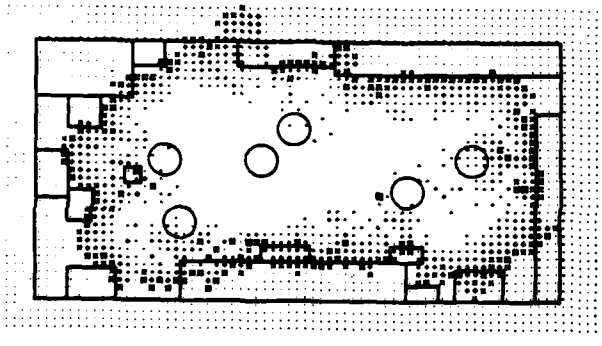


Figure 6: The Two-Dimensional Sonar Map After Thresholding.

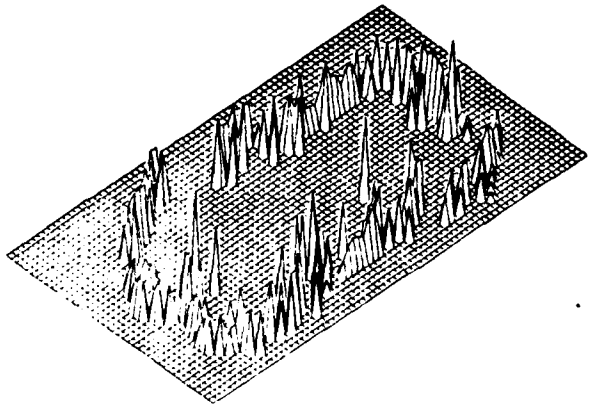


Figure 7: The Occupied Areas in the Sonar Map After Thresholding.

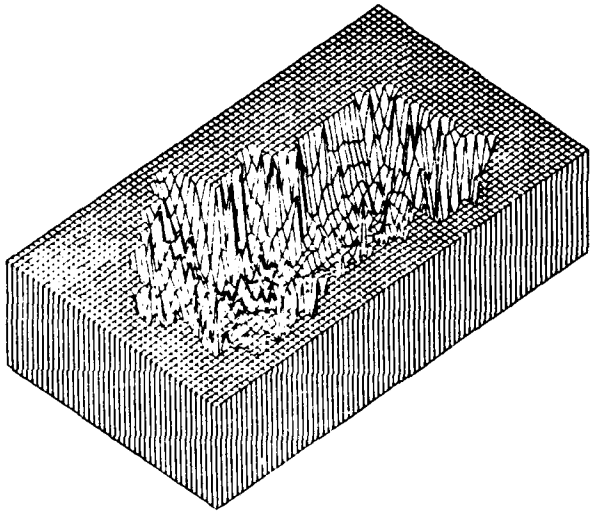


Figure 8: The Empty Areas in the Sonar Map After Thresholding.

A measure of the goodness of the match between two maps at a trial displacement and rotation is found by computing the sum of products of corresponding cells in the two maps. An occupied cell falling on an occupied cell contributes a positive increment to the sum, as does an empty cell falling on an empty cell (the product of two negatives). An empty cell falling on an occupied one reduces the sum, and any comparison involving an unknown value causes neither an increase nor a decrease. This naive approach is very slow. Applied to maps with a linear dimension of n, each trial position requires $O(n^2)$ multiplications. Each search dimension (two axes of displacement and one of rotation) requires $O(n)$ trial positions. The total cost of the approach thus grows as $O(n^5)$. With a typical n of 50 this approach can burn up a good fraction of an hour of Vax time.

Considerable savings come from the observation that most of the information in the maps is in the occupied cells alone. Typically only $O(n)$ cells in the map, corresponding to wall and object boundaries, are labelled occupied. A revised matching procedure compares maps A and B through trial transformation T (represented by a 2x2 rotation matrix and a 2 element displacement vector) by enumerating the occupied cells of A, transforming the co-ordinates of each such cell through T to find a

corresponding cell in B. The [A, B] pairs obtained this way are multiplied and summed, as in the original procedure. The occupied cells in B are enumerated and multiplied with corresponding cells in A, found by transforming the B co-ordinates through T^{-1} (the inverse function of T), and these products are also added to the sum. The result is normalized by dividing by the total number of terms. This procedure is implemented efficiently by preprocessing each sonar map to give both a raster representation and a linear list of the co-ordinates of occupied cells. The cost growns as $O(n^4)$, and the typical Vax running time is down to a few minutes.

A further speedup is achieved by generating a hierarchy of reduced resolution versions of each map. A coarser map is produced from a finer one by converting two by two subarrays of cells in the original into single cells of the reduction. Our existing programs assign the maximum value found in the subarray as the value of the result cell, thus preserving occupied cells. If the original array has dimension n, the first reduction is of size $n/2$, the second of $n/4$ and so on. A list of occupied cell locations is produced for each reduction level so that the matching method of the previous paragraph can be applied. The maximum number of reduction levels is $\log_2 n$. A match found at one level can be refined at the next finer level by trying only about three values of each of the two translational and one rotational parameters, in the vicinity of the values found at the coarser level, for a total of 27 trials. With a moderate a-priori constraint on the transformation this amount of search is adequate even at the first (coarsest) level. Since the cost of a trial evaluation is proportional to the dimension of the map, the coarse matches are inexpensive in any case. Applied to its fullest, this method brings the matching cost down to slightly larger than $O(n)$, and typical Vax times to under a second.

We found one further preprocessing step is required to make the matching process work in practice. Raw maps at standard resolutions (6 inch cells) produced from moderate numbers (about 100) of sonar measurements have narrow bands of cells labelled *occupied*. In separately generated maps of the same area the relative positions of these narrow bands shifts by as much as several pixels, making good registration of the occupied areas of the two maps impossible. This can be explained by saying that the high spatial frequency component of the position of the bands is noise, only the lower frequencies carry information. The problem is fixed by filtering (blurring) the occupied cells to remove the high frequency noise. Experiment suggests that a map made from 100 readings should be blurred with a spread of about two feet, while for map made from 200 readings a one foot smear is adequate. Blurring increases the number of cells labelled *occupied*. So as not to increase the computational cost from this effect, only the final raster version of the map is blurred. The occupied cell list used in the matching process is still made from the unfiltered raster.

With the process outlined here, maps with about 3000 six inch cells made from 200 well spaced readings can be matched with an accuracy of about six inches displacement and three degrees rotation in one second of Vax time.

5. Results

We incorporated the sonar map builder into a system that successfully navigates the Neptune robot through cluttered obstacle courses. The existing program incrementally builds a single sonar map by combining the readings from successive vehicle stops made about one meter apart. Navigation is by dead reckoning - we do not yet use the sonar map matching code. The next move is planned in the most recent version of the map by a path-planning method based on path relaxation [11]. Since this method can cope with a probabilistic representation of occupied and empty areas and does path-planning in a grid, it fits naturally into our present framework. The system has successfully driven Neptune the length of our cluttered 30 by 15 foot laboratory using less than one minute of computer time.

6. Conclusions

We have described a program that builds moderately high resolution spatial maps of a mobile robot's surroundings by combining several hundred range readings from unadorned Polaroid ultrasonic units. The main innovation is an efficient mathematical method that reduces the position uncertainty of objects detected by wide angle sonar beams by combining interlocking constraints in a raster occupation probability map. We have also developed a fast algorithm for relating two maps of the same area to derive relative displacement, angle and goodness of match.

We have used this mapping method in a system that navigates a mobile robot to a desired destination through obstacles and clutter, and are preparing a more elaborate navigation system that depends on matching of the sonar maps to recognize key locations and on higher-level representations to navigate over long distances.

7. Acknowledgments

The authors would like to thank Gregg W. Podnar for his help in using the Neptune robot and performing some of the experiments, Michael Fuhrman for precious advice on the use of a 3D drawing package, and Larry Matthies for providing the means to print the corresponding drawings.

References

1. Baggeroer, A.B. Sonar Signal Processing. In *Applications of Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, N.J., 1978.

2. Chattergy, A. Some Heuristics for the Navigation of a Robot. Robotics Research Laboratory, Department of Electrical Engineering, University of Hawaii, 1984.

3. Crowley, J.L. Position Estimation for an Intelligent Mobile Robot. 1983 Annual Research Review, The Robotics Insutute, Carnegie-Mellon University, Pittsburgh, PA, 1984.

4. Hussey, M.. *Diagnostic Ultrasound: An Introduction to the Interactions between Ultrasound and Biological Tissues*. Blackie & Son Limited, London, 1975.

5. Matthies, L.H. and Thorpe, C.E. Experience With Visual Robot Navigation. Proceedings of IEEE Oceans 84, IEEE , Washington, D.C., August, 1984.

6. Miller, D. Two Dimensional Mobile Robot Positioning Using Onboard Sonar. Pecora IX Remote Sensing Symposium Proceedings, IEEE, Sioux Falls, SD, October, 1984.

7. Moravec, H.P. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. Ph.D. Th., Stanford University, May 1980. Also available as Stanford AIM-340, CS-80-813 and CMU-RI-TR-3

8. Podnar, G.W., Blackwell, M.K. and Dowling, K. A Functional Vehicle for Autonomous Mobile Robot Research. CMU Robotics Institute, April, 1984.

9. Polaroid Corporation. *Ultrasonic Range Finders*. Polaroid Corporation, 1982.

10. Thorpe, C.E. The CMU Rover and the FIDO Vision and Navigation System. Symposium on Autonomous Underwater Robots, University of New Hampshire, Marine Systems Engineering Lab, May, 1983.

11. Thorpe, C.E. Path Relaxation: Path Planning for a Mobile Robot. Technical Report CMU-RI-TR-84-5, CMU Robotics Institute, April, 1984. Also in Proceedings of IEEE Oceans 84, Washington, D.C., August, 1984 and Proceedings of AAAI-84, Austin, Texas, August, 1984

A Sonar-Based Mapping and Navigation System

Alberto Elfes

The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

This paper describes a sonar-based mapping and navigation system for autonomous mobile robots operating in unknown and unstructured surroundings. The system uses sonar range data to build a multi-leveled description of the robot's environment. Sonar maps are represented in the system along several dimensions: the Abstraction *axis, the* Geographical *axis, and the* Resolution *axis. Various kinds of problem-solving activities can be performed and different levels of performance can be achieved by operating with these multiple representations of maps. The major modules of the* Dolphin *system are described and related to the various mapping representations used. Results from actual runs are presented, and further research is mentioned. The system is also situated within the wider context of developing an advanced software architecture for autonomous mobile robots.*

1. Introduction

The Dolphin system is intended to provide sonar-based mapping and navigation for an autonomous mobile robot operating in unknown and unstructured environments. The system is completely autonomous in the sense that it has no *a priori* model or knowledge of its surroundings and also carries no user-provided map. It acquires data from the real world through a set of sonar sensors and uses the interpreted data to build a multi-leveled and multi-faceted description of the robot's operating environment. This description is used to plan safe paths and navigate the vehicle towards a given goal.

The system is intended for indoor as well as outdoor use; it may be coupled to other systems, such as vision, to locate landmarks that would serve as intermediate or final destinations.

In the course of this paper, we will briefly identify some of the conceptual processing levels needed for mobile robot software, relate the present system to this framework, discuss the multiple representations developed for sonar maps as well as their use in different kinds of problem-solving activities, describe the overall system architecture and show some results from actual runs. We finish with an outline of further research.

2. Conceptual Processing Levels for an Autonomous Mobile Robot

The sonar mapping and navigation system discussed here is part of a research effort that investigates various issues involved in the development of the software structure of an autonomous mobile robot. To situate the Dolphin system within this wider context, we characterize in this section some of the conceptual processing levels required for an autonomous vehicle (see Fig. 2-1). Each is briefly discussed below:

VII. Global Control
VI. Global Planning
V. Navigation
IV. Real-World Modelling
III. Sensor Integration
II. Sensor Interpretation
I. Robot Control

Figure 2-1: Conceptual Activity Levels in a Mobile Robot Software Architecture.

- *Robot Control:* This level takes care of the physical control of the different sensors and actuators available to the robot. It provides a set of primitives for locomotion, actuator and sensor control, data acquisition, etc., that serve as the robot interface, freeing the higher levels of the system from low-level details. This would include dead-reckoning motion estimation and monitoring of internal sensors. *Internal Sensors* provide information on the status of the different physical subsystems of the robot, while *External Sensors* are used to acquire data from the robot's environment.
- *Sensor Interpretation:* On this level the acquisition of sensor data and its interpretation by Sensor Modules is done. Each Sensor Module is specialized in one type of sensor or even in extracting a specific kind of information from the sensor data. They provide information to the higher levels using a common representation and a common frame of reference.
- *Sensor Integration:* Due to the intrinsic limitations of any sensory device, it is essential to integrate information coming from qualitatively different sensors. Specific assertions provided by the Sensor Modules are correlated to each other on this level. For example, geometric boundaries of an obstacle extracted by sonar can be projected onto an image provided by the vision subsystem and can help in identifying

a certain object. On this level, information is aggregated and assertions about specific portions of the environment can be made.

- *Real-World Modelling:* To achieve any substantial degree of autonomy, a robot system must have an understanding of its surroundings, by acquiring and manipulating a rich model of its environment of operation. This model is based on assertions integrated from the various sensors, and reflects the data acquired and the hypotheses proposed so far. On this level, local pieces of information are used in the incremental construction of a coherent global Real-World Model; this Model can then be used for several other activities, such as landmark recognition, matching of newly acquired information against already stored maps, and generation of expectancies and goals.
- *Navigation:* For autonomous locomotion, a variety of problem-solving activities are necessary, such as short-term and long-term path-planning, obstacle-avoidance, detection of emergencies, etc. These different activities are performed by modules that provide specific services.
- *Global Planning:* To achieve a global goal proposed to the robot, this level provides task-level planning for autonomous generation of sequences of actuator, sensor and processing actions. Other necessary activities include simulation, error detection, diagnosis and recovery, and replanning in the case of unexpected situations or failures.
- *Global Control:* Finally, on this level Supervisory Modules are responsible for the scheduling of different activities and for combining Plan-driven with Data-driven activities in an integrated manner so as to achieve coherent behaviour.

This conceptual structure provides a paradigm within which several of our research efforts are situated [6, 11, 12]. It has influenced, in particular, the architecture of the Dolphin system for sonar-based mapping and navigation, as mentioned in Section 5.

3. Sonar Mapping

3.1. Introduction

The Dolphin sonar system is able to build dense maps of the robot's environment and use them for autonomous navigation. The central representation of sonar mapping information is the *Probabilistic* or *Sensor-Level Local Map*, which uses a medium-resolution grid (with a typical accuracy of 0.5 ft). The cells of a two-dimensional array spanning the area of interest are used to store occupancy information (EMPTY, OCCUPIED or UNKNOWN), as well as the associated confidence factors.

Currently, the cycle of operation of the sonar system is as follows: from its current position, the robot acquires a set of range measurements provided by the sonar sensor array; these readings are then interpreted as assertions concerning *empty* and *occupied* areas, and serve to update the sonar map. The map is now used to plan a safe path around obstacles, and the robot moves a certain distance along the path. It updates its position and orientation estimate and repeats the cycle.

3.2. Building Maps

The Local Map building process is discussed in detail in [11], and is reviewed here only briefly. We proceed to describe how other representations are derived from it.

The sonar sensor array is composed of 24 Polaroid laboratory grade ultrasonic transducers. These devices are arranged in a ring and controlled by a microprocessor that also interfaces to a VAX mainframe. For experimental runs, the array was mounted on two different robots (*Neptune* [13] for indoor runs, and the *Terragator* [12] for outdoors).

The mapping system processes range measurements obtained from the sonar transducers, annotated with the positions of the corresponding sensors, which are derived from the position and orientation of the robot. Each measurement provides information about *probably empty* and *possibly occupied* volumes in the space subtended by the beam (a 30° cone for the present sensors). This occupancy information is projected onto a rasterized two-dimensional horizontal map. Sets of readings taken both from different sensors and from different positions of the robot are incrementally integrated into the sonar map, using a probabilistic approach. In this way, errors and uncertainties are reduced and the map becomes gradually more detailed.

The sonar beam is modelled by probability distribution functions. Informally, these functions describe our confidence that the points inside the cone of the beam are empty and our uncertainty about the location of the point that caused the echo. The functions are based on the range value and on the spatial sensitivity pattern of the sonar device.

These sonar maps are very useful for motion planning. They are much denser than those made by typical stereo vision programs, and computationally at least one order of magnitude faster to produce.

3.3. Related Work

In the Robotics area, ultrasonic range transducers have recently attracted increasing attention. This is due in part to their simplicity, low cost and the fact that distance measurements are provided directly. Some research has focused specifically on the development of more elaborate beam-forming and detection devices (see, for example, [8]), or on the application of highly sophisticated signal processing techniques [1] to complex sonar signals.

Specific applications of sonar sensors in robot navigation include determining the position of a robot given a known map of the environment [9, 10, 5] and some *ad hoc* navigation schemes [2]. An independent CMU sonar mapping and navigation effort [3, 4] uses a narrow beam, formed by a parabolic reflector, to build a line-based description of the environment.

4. Multiple Axis of Representation of Sonar Mapping Information

From the Probabilistic Local Maps described in the previous section, several other data structures are derived. We use the following dimensions of representation (Fig. 4-1):

- THE ABSTRACTION AXIS: Along this axis we move from a sensor-based, data-intensive representation to increasingly higher levels of interpretation and abstraction. Three levels are defined: the *Sensor Level*, the *Geometric Level* and the *Symbolic Level*.
- THE GEOGRAPHICAL AXIS: Along this axis we define *Views*, *Local Maps* and *Global Maps*, depending on the extent and characteristics of the area covered.
- THE RESOLUTION AXIS: Sonar Maps are generated at different values of grid resolution for different applications. Some computations can be performed satisfactorily at low levels of detail, while others need higher or even multiple degrees of resolution.

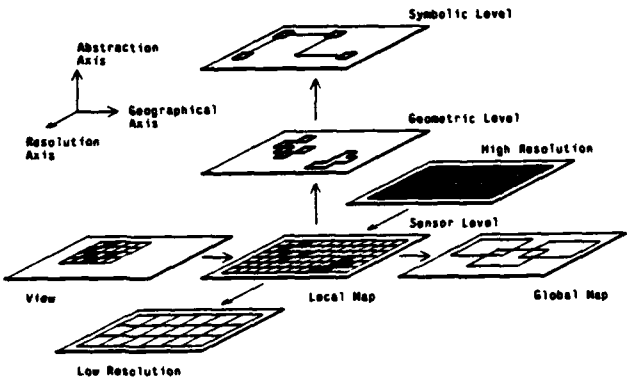


Figure 4-1: Multiple Axis of Representation of Sonar Maps.

4.1. The Abstraction Axis

The first kind of sonar map built from the sonar range data uses the *Probabilistic* representation described earlier. A two-dimensional grid covering a limited area of interest is used. This map is derived directly from the interpretation of the sensor readings and is, in a sense, the description closest to the real world. It serves as the basis from which other kinds of representations are derived. Along the Abstraction Axis, this data-intensive description is also defined as the *Sensor Level* Map.

The next level is called the *Geometric Level*. It is built by scanning the Sensor Level Map and identifying blobs of cells with high OCCUPIED confidence factors. These are merged into uniquely labeled objects with explicitly represented polygonal boundaries. If needed, the same can be done with EMPTY areas.

The third is the *Symbolic Level*, where maps of larger areas (typically Global Maps) are described using a graph-like representation. This description bears only a topological equivalence to the real world. Nodes represent "interesting" areas, where more detailed mapping information is necessary or available, while edges correspond to simpler or "uninteresting" areas (navigationally speaking), such as corridors.

Different kinds of problem-solving activities are better performed on different levels of abstraction. For example, global path-planning (such as how to get from one building wing to another) would be done on the symbolic level, while navigation through a specific office or lab uses the sensor-level map, where all the detailed information about objects and free space, as well as the associated certainty factors, is stored.

4.2. The Geographical Axis

In order to be able to focus on specific geographical areas and to handle portions of as well as complete maps, we define a hierarchy of maps with increasing degrees of coverage. Progressing along the Geographical Axis, we start with *Views*, which are maps generated from scans taken from the current position, and that describe the area visible to the robot from that place. As the vehicle moves, several Views are acquired and integrated into a *Local Map*. The latter corresponds to physically delimited spaces such as labs or offices, which define a connected region of visibility. *Global Maps* are sets of several Local Maps, and cover wider spaces such as a whole wing of a building, with labs, offices, open areas, corridors, etc.

4.3. The Resolution Axis

Finally, along the Resolution Axis, we again start with the Sensor-Level Local Map and generate a progression of maps with increasingly less detail. This allows certain kinds of computations to be performed either at lower levels of resolution with correspondingly less computational expense, or else enables operations at coarser levels to guide the problem-solving activities at finer levels of resolution.

The most detailed sonar maps that can be obtained from the method outlined in Section 3 (considering the intrinsic limitations of the sensors) have a cell size of 0.1 × 0.1 ft . For navigation purposes, we have typically been using a 0.5 ft grid for indoors and a 1.0 ft grid for outdoors. Nevertheless, several operations on the maps are expensive and are done more quickly at even lower levels of resolution. For these cases we reduce higher resolution maps by an averaging process that produces a coarser description. One example of an application of this technique is the Map Matching procedure described in [11], where two Local Maps being compared with each other are first matched at a low level of detail. The result then constrains the search for a match at the next higher level of resolution.

5. Overall System Architecture

To provide a context for these multiple descriptions, we present in this Section the overall architecture of the Dolphin Sonar-Based Mapping and Navigation system (Fig. 5-1). The function of the major modules and their interaction with the various sonar map representations [7] is described below:

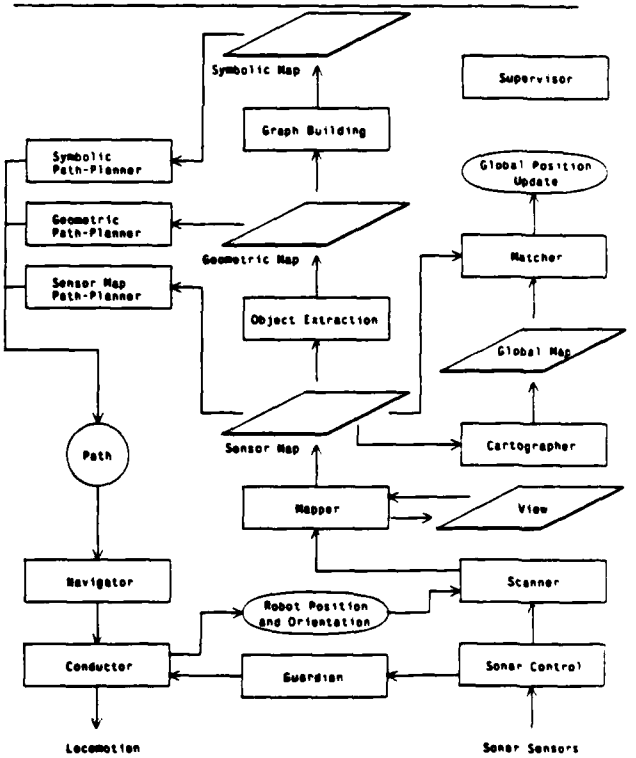


Figure 5-1: Architecture of the Sonar Mapping and Navigation System.

Sonar Control: Interfaces to and runs the sonar sensor array, providing range readings.

Scanner: Preprocesses and filters the sonar data. Annotates it with the position and orientation of the corresponding sensor, based on the robot's motion estimate.

Mapper: Using the information provided by the Scanner, generates a View obtained from the current position of the robot. This View is then integrated into a Local Map.

Cartographer: Aggregates sets of Local Maps into Global Maps. Provides map handling and bookkeeping functions.

Matcher: Matches a newly acquired Local Map against portions of Global Maps for operations such as landmark identification or update of the absolute position estimate.

Object Extraction: Obtains geometric information about obstacles. Objects are extracted by merging blobs of OCCUPIED cells and determining the corresponding polygonal boundaries. A region-coloring approach is used for unique labeling.

Graph Building: Searches for larger regions that are either empty or else have complex patterns of obstacles, labeling them as "free" or "interesting" spaces.

Path-Planning: Three levels of path-planning are possible: *Symbolic Path-Planning* is done over wider areas (Global Maps) and at a higher level of abstraction (Symbolic Maps); *Geometric Path-Planning* is done as an intermediary stage, when the uncertainty in Local Maps is low; and *Sensor Map Path-Planning* is used to generate detailed safe paths. The latter performs an A* search over the map cells, with the cost function taking into account the obstacle certainty factors and the distance to the goal. The planned path is provided to the Navigator.

Navigator: Takes care of the overall navigation issues for the vehicle. This includes examining already planned paths to determine whether they are still usable, invoking the path-planner to provide new paths, setting intermediary goals, overseeing the actual locomotion, etc.

Conductor: Controls the physical locomotion of the robot along the planned path. The latter is currently approximated by sequences of line segments, using a line-fitting approach. Provides an estimate of the new position and orientation of the robot.

Guardian: During actual locomotion, this module checks the incoming sonar readings and signals a stop if the robot is coming too close to a (possibly moving) obstacle not detected previously. It serves as a "sonar bumper".

Supervisor: Oversees the operation of the various modules and takes care of the overall control of the system. It also provides a user interface.

Comparing this architecture with the activities outlined in Section 2, we see that the Sonar Control and Conductor modules belong to the Robot Control level; the Scanning and Mapping modules operate on the Sensor Interpretation level; the Object Extraction, Graph Building, Cartographer and Matcher modules provide functions on the Real-World Modelling level; Path-Planning, the Guardian and Navigation are situated on the Navigation level; and the Supervisor belongs to the Control level.

6. Tests of the System

The Dolphin system described here was tested in several indoor runs in cluttered environments using the *Neptune* mobile robot [13], developed at the Mobile Robot Laboratory of the Robotics Institute, CMU. It was also tested in outdoor environments, operating among trees, using the *Terragator* robot in the context of the CMU ALV project. The system operated successfully in both kinds of environments, navigating the robot towards a given destination.

In Fig. 6-1, an example run is given. The sequence of maps presented shows how the sonar map becomes gradually more detailed and how the path is improved, as more information is gathered. The example corresponds to an indoor run, done in our laboratory. A distance of approximately 25 ft was covered; the grid size is 0.5 ft. Objects present in the lab included chairs, tables, boxes, workstations, filing cabinets, etc. *Empty* spaces with high certainty factors are represented by white areas; lower certainty factors by "." symbols of increasing thickness. *Occupied* areas are shown using "×" symbols, and *Unknown* areas using "·". The planned path is shown as a dotted line, and the route actually followed by the robot as solid line segments. The starting point is a solid + and the goal a solid ×.

In Fig. 6-2, an outdoor run is shown, together with an example of the Object Extraction algorithm. The objects are uniquely identified and the polygonal boundaries are shown. The map corresponds to a run done among trees. A distance of approximately 50 ft was traversed. The grid size was 1.0 ft, which proved adequate for navigation, but did not allow a more precise description of the real boundaries of the detected objects.

7. Further Research

We conclude our discussion by outlining in this Section some research lines to be further pursued.

7.1. Handling Position Uncertainty

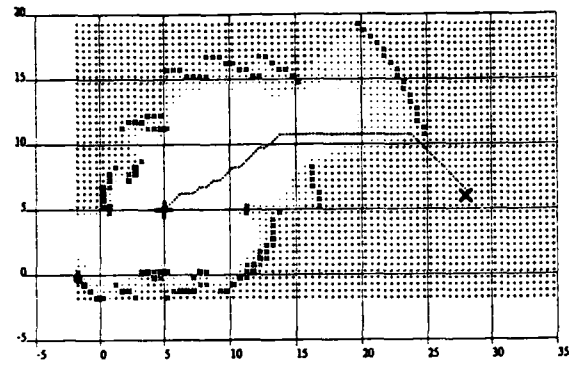
Our current system presupposes that the position and orientation of the robot (and by that, of the sonar sensors) as it acquires sonar data is known with reasonable precision. This is crucial for integrating readings taken over shorter distances, which are combined as previously outlined. Drifts over longer distances are inevitable, but lead only to a topological distortion of the map.

To update the current position of the robot, we presently rely on dead-reckoning estimates based on wheel encoders and an onboard inertial navigation system. These drift with travelling time and distance. As a result, ground truth (the real-world environment) and the sonar map drift apart. This problem is characteristic of navigation without access to absolute position information. In stereo vision navigation, it has traditionally been addressed by estimating motion based on image matching.

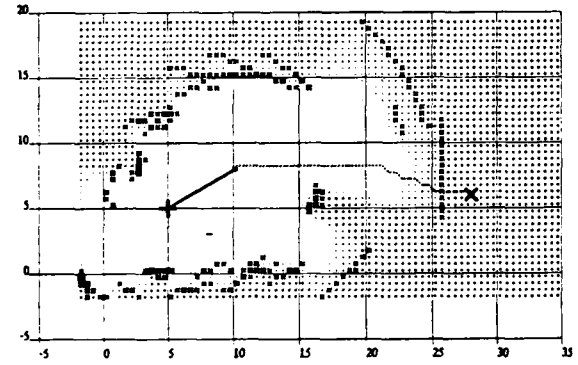
We are currently investigating two complementary approaches to this problem: incorporating the uncertainty in the position of the robot into the map-making process and do motion solving by matching new sets of readings against the map being incrementally built.

7.2. Extending the Architecture

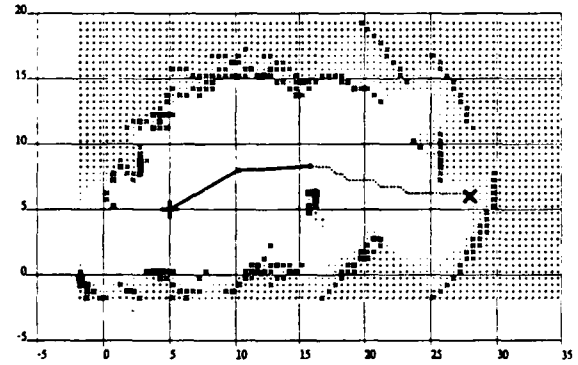
The architecture described above embodies a sequential control-flow organization. This, however, does not reflect the problem-solving characteristics inherent to mobile robot software. The various modules involved in the problem-solving effort are frequently quasi-independent and have a low degree of coupling; therefore, they should conceptually proceed in parallel, interacting with each other as needed. We have recently started the implementation of a distributed version of Dolphin [12] along the lines discussed in [6], where multiple agents work on concurrent activities.



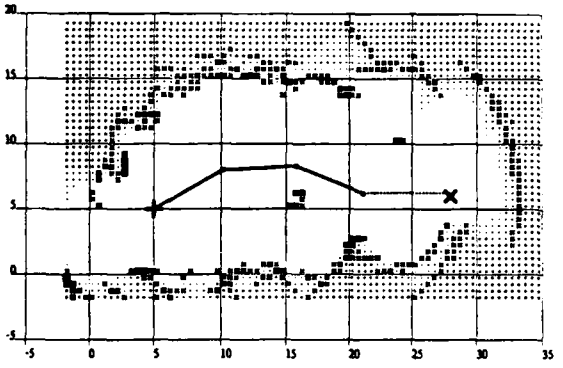
(a)



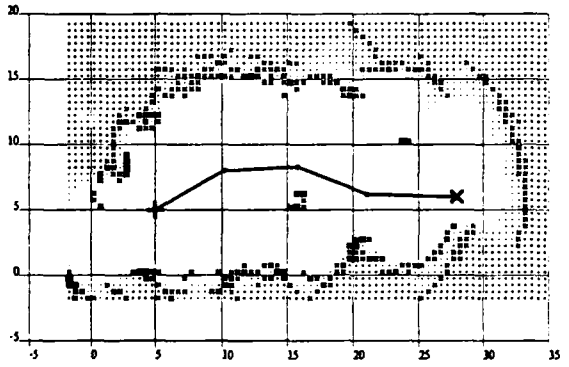
(b)



(c)



(d)



(e)

Figure 6-1: An Example Run. This run was performed indoors, in the Mobile Robot Lab. Distances are in ft. Grid size is 0.5 ft.

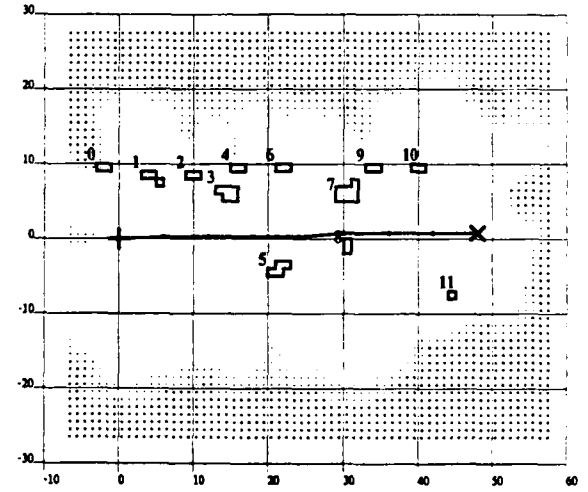


Figure 6-2: Objects Extracted from a Sonar Map. The objects are numbered and their polygonal boundaries are shown. This map describes an outdoor run, and the objects are trees. Distances are in ft. Grid size is 1.0 ft.

Another issue we are currently investigating is the development of a task-level Global Planner that would automatically generate a Control Plan, establishing sequences of parallel and sequential actions. We are considering a hierarchical approach similar to NOAH [14], using a graph to represent the plan and explicitly storing alternatives and sensor-dependent conditions as part of it. The elementary operations of sensor information gathering, interpretation, actuator control and specific problem-solving activities are the primitives used by the planner.

8. Conclusions

We have described a system that uses a Sensor Level, probability-based sonar map representation of medium resolution to build several kinds of maps. Three different dimensions of representation are defined: the Abstraction Axis, the Geographical Axis and the Resolution Axis. These maps are used by a sonar mapping and navigation system that performed successfully in indoor and outdoor environments. We are now investigating motion recovery techniques and expanding the system to test distributed control and global planning mechanisms.

9. Acknowledgments

The author would like to thank Hans P. Moravec for his support and for providing several important insights into the topics discussed in this paper. Gregg W. Podnar for his help with the Neptune robot, Chuck Thorpe for the initial code for the path-planner, and Richard Redpath for providing more reliable communication to the sonar ring, as well as for assistance during outdoor runs.

This research is being supported in part by the Office of Naval Research under Contract N00014-81-K-0503, in part by Denning Mobile Robotics, Inc., and in part by the Western Pennsylvania Advanced Technology Center. The author is supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, Brazil, under Grant 200.986-80, in part by the Instituto Tecnológico de Aeronáutica - ITA, Brazil, and in part by The Robotics Institute, Carnegie-Mellon University.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the funding agencies.

References

[1] Baggeroer, A.B.
Sonar Signal Processing.
Signal Processing Series. Applications of Digital Signal Processing.
Prentice-Hall, Englewood Cliffs, N.J., 1978.

[2] Chattergy, A.
Some Heuristics for the Navigation of a Robot.
Technical Report, Robotics Research Laboratory, Department of Electrical Engineering, University of Hawaii, 1984.

[3] Crowley, J.L.
Position Estimation for an Intelligent Mobile Robot.
In *1983 Annual Research Review*. The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 1984.

[4] Crowley, J.L.
Dynamic World Modelling for an Intelligent Mobile Robot Using a Rotating Ultra-Sonic Ranging Device.
In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*. IEEE, St. Louis, Missouri, March, 1985.

[5] Drumheller, M.
Mobile Robot Localization Using Sonar.
Technical Report AI-M-826, Artificial Intelligence Lab, Massachusetts Institute of Technology, January, 1985.

[6] Elfes, A. and Talukdar, S.N.
A Distributed Control System for the CMU Rover.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence - IJCAI-83*. IJCAI, Karlsruhe,Germany, August, 1983.

[7] Elfes, A.
Multiple Levels of Representation and Problem-Solving Using Maps From Sonar Data.
In Weisbin, C.R. (editor), *Proceedings of the DOE/CESAR Workshop on Planning and Sensing for Autonomous Navigation*. Oak Ridge National Laboratory, UCLA, Los Angeles, August 18-19,, 1985.
Invited Paper.

[8] Miller, G.L., Boie, R.A. and Sibilia, M.J.
Active Damping of Ultrasonic Transducers for Robotic Applications.
In *Proceedings of the International Conference on Robotics*. IEEE, Atlanta, Georgia, March, 1984.

[9] Miller, D.
Two Dimensional Mobile Robot Positioning Using Onboard Sonar.
In *Pecora IX Remote Sensing Symposium Proceedings*. IEEE, Sioux Falls, SD, October, 1984.

[10] Miller, D.
A Spatial Representation System for Mobile Robots.
In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*. IEEE, St. Louis, Missouri, March, 1985.

[11] Moravec, H.P. and Elfes, A.
High-Resolution Maps from Wide-Angle Sonar.
In *IEEE International Conference on Robotics and Automation*. IEEE, March, 1985.

[12] Moravec, H.P. et al.
Towards Autonomous Vehicles.
1985 Robotics Research Review.
The Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[13] Podnar, G.W., Blackwell, M.K. and Dowling, K.
A Functional Vehicle for Autonomous Mobile Robot Research.
Technical Report CMU-RI-TR-84-28, The Robotics Institute, Carnegie-Mellon University, April, 1984.

[14] Sacerdoti, E.D.
A Structure for Plans and Behavior.
Elsevier, New York, N.Y., 1977.

Three Dimensional Images from Cheap Sonar

Hans P. Moravec
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

December 21, 1985

1 Introduction

We propose to build and use moderate resolution three dimensional space occupancy maps built from multiple measurements from cheap sonar sensors. By cheap sonar I mean range readings obtained from unmodified Polaroid sonar transducers driven by the original Polaroid circuit board, or by an improved board (allowing closer minimun ranges) from Texas Instruments. This is a simple, but highly developed and reliable, not to mention inexpensive, system that returns the distance to the nearest reflector in a certain wide cone of sensitivity. Though much more information can be obtained, in principle, from single sound bursts by modifying the aperture, phase relationships, frequencies and processing, such an approach ignores the present very good solution.

2 Past Work

In earlier work [Moravec&Elfes 1985] we described the use of multiple wide-angle sonar range measurements to map the surroundings of an autonomous mobile robot. A sonar range reading provides information concerning empty and occupied volumes in a cone (subtending 30 degrees in our case) in front of the sensor. The reading is modelled as probability profiles projected onto a rasterized map, where somewhere occupied and everywhere empty areas are represented. Range measurements from multiple points of view (taken from multiple sensors on the robot, and from the same sensors after robot moves) are systematically integrated in the map. Overlapping empty volumes re-inforce each other, and serve to condense the range of occupied volumes. The map definition improves as more readings are added. The final map shows regions probably occupied, probably unoccupied, and unknown areas. The method deals effectively with clutter, and can be used for motion planning and for extended landmark recognition. This system was tested on our *Neptune* mobile robot, and recently outdoors on the *Terregator* robot.

3 Experimental Approach

Processing a single reading from a standard unit is computationally cheap; only one number is generated, limiting the computations necessary or possible. The range accuracy of a typical reading is better than a centimeter, but because of the wide angle of the pulse, the lateral position of the reflection is uncertain to on the order of a meter. By exercising multiple units repeatedly, readings from multiple viewpoints may be combined to deduce the location of the reflecting surfaces more precisely. The combining process is a kind of deconvolution - each point in the final high resolution map is a consequence of many of the individual readings combined in a particular, unique way and each reading participates in many map points.

Our existing approach uses the idea that the interior of each sonar reading cone (bounded by the sensitivity profile laterally, and by the range surface lengthwise) is known to be empty, and that the reflecting point is somewhere on the range surface in this cone. The empty interiors of other readings overlapping this range surface reduce the region of uncertainty of the location of the echoing point in a probabilistic way, while intersecting range surfaces reinforce each other at the intersections. The deconvolution is essentially non-linear.

The old programs work in two dimensions, collapsing the measurement cones vertically into flat pie wedges that are combined in a two dimensional map array that ultimately holds numbers giving the confidence that a given cell is empty or occupied. We have experimentally noted that maps with a range of 10 meters and a resolution of 15 to 30 cm can be reliably constructed with data from a ring of 24 robot-mounted transducers looking out horizontally at 15 degree intervals and pulsed at six locations a few meters apart in the robot's travels (144 independent measurements). The sharpness of the map can be seen to improve as more readings are added. Many readings are combined to form one map probability point, and this process makes our method quite tolerant to the occasional range errors encountered in the sonar data.

A highly optimized version of the program, using fixed point arithmetic, can process 144 points in roughly 1 second on a big Vax, 2 seconds on a Sun2 and 4 seconds on a Macintosh, building a 32x32 map of eight bit probabilities. A companion program correlates two such maps, using a coarse to fine hierarchy of reductions and a dual representation (raster and list of occupied cells) to search over X, Y shift and angle, in similar times. Another program devises good robot paths through the probability maps.

3.1 3D mapping

Our approach generalizes very naturally to three dimensions - in fact the collapse of cones to wedges in the 2D program is its greatest single approximation, and information waster.

The sensors must be configured differently, however. The only height information in the present planar ring comes from the vertical divergence of the cones of sensitivity, whose symmetry makes it impossible in principle to distinguish reflections from above the ring plane from those an equal distance below the plane. Even without this ambiguity, the present arrangement could provide very little vertical resolution.

An arrangement of sensors on the surface of a partial sphere would be much better. The 15 degree spacing of the 24 sensors on the planar ring was chosen to give some overlap of fields of view. It was discovered that this spacing allowed multiple sensors to be fired simultaneously without serious interference, in three, or even two, interleaved banks, greatly speeding data gathering. Using the same idea and spacing to fill a sphere instead of a circle leads to the following calculation.

A sphere represents 4π of solid angle. Spacing the sensors 15 degrees from each other assigns a cone with 15 degree apex to each sensor. A cone with apex angle T subtends $2\pi(1-\cos(T/2))$ solid angle, and we can (glossing over packing problems) arrange about $4\pi/(2\pi(1-\cos(T/2))) = 2/(1-\cos(T/2))$ of them

into a sphere. With T=15 degrees 233 transducers fill a sphere. If we content ourselves with a 90 degree wedge (almost a fisheye if you note that the beams fan out an additional 15 degrees on all edges, for a net coverage of 120 degrees) then this gets reduced to a more manageable 34 transducers.

If actually packed onto a spherical cap, the sensor group would greatly resemble a compound insect eye, each facet being a Polaroid transducer. The insect would be a monster. The transducers are somewhat less than 5cm in diameter, which would demand a sphere radius of about 40cm. A 90 degree cap from this sphere would be a shallow bowl 56cm in diameter and 12cm deep.

One such sensor array on the nose of a vehicle, tilted down somewhat, should be adequate for many tasks, but imagine getting better side coverage, say for obstacle avoidance, by placing two, one on each side of the head, enhancing the giant insect effect.

3.2 How Many Readings, How much Computation?

The 3D map we hope to derive from this array has more cells than the 2D maps we have worked with, and will require more data. How much?

Suppose we build our maps to a range of about 10 meters in the vehicle forward direction, 5 meters laterally and 3 meters in the vertical direction, and to a resolution of 30cm in each direction. There will be 33x17x10 cells, each holding a number, in the final map. This is 5,610 numbers. A naive degrees of freedom analysis suggests that a similar number of readings each returning one number are necessary to *determine this many variables.* Fortunately our 2D experience suggests that far fewer will suffice.

We have noted experimentally that 144 readings nicely spaced around our cluttered laboratory is just enough to give us good 32 cell by 32 cell maps covering a square area 10 meters on a side. There are 1024 points in such maps, so we seem to be accomplishing the impossible, extracting 1024 unknowns from 144 equations. Actually, the 1024 numbers are not very informative as their magnitude represents our certainty (or uncertainty) about particular cells being occupied, not something intrinsic about the scenery. Most of the cells in the final mape are labelled an unsurprising "unknown" (represented by 0) or "probably empty" (represented by a negative number). The real information is concentrated in the locations of the reflecting boundary seen by the robot, i.e. the minority of cells labelled "probably *occupied*". *To first approximation this boundary is a one dimensional contour embedded in the 2D map.* Its length in cells is on the order of the boundary length of the map, 4x32. The information is not in the contents of these cells (positive probability numbers), but in their location. Each cell represents about one number - think of the boundary expressed in polar co-ordinates - the information is in the radius at each angle, the angle itself is just the independent variable. SO - we have 144 equations to determine about 4x32 = 128 variables - just about right! Mathematics is great.

In 3D the contour becomes a surface. In our example of two paragraphs ago the map size was 33x17x10 cells. The surface of this volume has about 2,100 cells, and thus requires about 2,100 readings by the above analysis, or 62 full scans of the 34 transducers in the 90 degree eye. The sensors

can be pulsed about twice per second. With two way interleaving, a full eye poll takes a second. The 62 readings would thus take about a minute. Computation times on a big Vax, extrapolating from the fast 2D program, would also be at about 30 seconds to a minute. It is assumed that the robot travels about ten meters during this minute (a speed of 0.6 km/hr) to give each reading set a fresh vantage point, and that adequate dead reckoning is provided to correctly relate the 60 sets geometrically. Of course, lower resolution maps, or simple obstacle detection, can be accomplished faster, in as little as one (half second) pulse gathering period.

These numbers suggest that high speed travel is best left to longer range sensors, and perhaps simpler techniques. The sonar mapping could be very useful for slow, close in, tight maneuvering in complicated environments and on very rough ground. The very general path passability grid route planners demonstrated by the group extend in a natural way to the dense 3D data this approach will provide.

4 Research Plan

All our sonar experiments so far have been conducted with early prototype sonar rings provided by our sometime collaborator, Denning Mobile Robotics, Inc. of Woburn, Massachusetts. Because of a rather old fashioned (small buffer) serial interface on our Vax computers, the processors on these rings can't reliably communicate with the Vaxes in the present configuration, and this has been a serious hinderance to sonar experimentation. We will begin the work by building new interfaces for the transducers using Texas Instrument driver boards funneling into an MC68000 microprocessor. Denning has agreed to help in this effort - they have been using a TI board based design successfully for six months.

A second stage is design and construction of the physical array. This will require a mathematical optimization and an evaluation by simulations of the individual sensor placements.

The bulk, and point, of the work will be an extended series of experiments with 3D map building and navigation programs. One small but interesting subproblem in the early stages is 3D raster fill of conically bounded sphere surfaces and volumes. A more significant problem is the handling of position uncertainty in the measurements made during an extended run. Our probability raster permits a very direct representation for uncertainty - it can simply be added to the probability distribution, increasing the spread of each reading in the appropriate directions.

We'd like to try an approach that projects the incremental uncertainty of each move onto old measurements rather than new ones. The result would be a map that is always detailed for the local area around the vehicle, and fades to fuzziness under the cumulative effect of errors in the distance. Very old readings that provide almost no information because of uncertainty in their location could eventually be eliminated from the mapmaking.

The three dimensional nature of the images will permit some work in identification of large objects. Recognition of small objects is ruled out by the coarseness (about 10cm) of the anticipated maps.

Visual Navigation

Experiments and Thoughts on Visual Navigation

C. Thorpe, L. Matthies, and H. Moravec

Carnegie-Mellon University

Abstract

We describe a second generation system that drives a camera-equipped mobile robot through obstacle courses. The system, which evolved from earlier work by Moravec [6], incorporates a new path planner and has supported experiments with interest operators, motion estimation algorithms, search constraints, and speed-up methods. In this paper we concentrate on the effects of constraint and on speed improvement. We also indicate some of our plans for a follow-on system.

1. Introduction

FIDO is a navigation and vision system for a robot rover. Using only stereo vision, it locates obstacles, plans a path around them, and tracks the motion of the robot as it moves. FIDO's main loop repeatedly:

- picks about 40 points from one member of a stereo *image pair*
- stereo-ranges those points by a hierarchical correlation technique
- plans a path that avoids those points
- moves forward
- takes two new stereo pictures
- relocates those same points and stereo ranges them again
- deduces vehicle motion from apparent point motion.

This paper describes our experimental investigations and improvements in FIDO's performance. Early versions of FIDO and its predecessor, the Stanford Cart programs, used 9-eyed stereo, took 15 minutes or more per step, and were not always reliable. By using additional geometric constraints, we have been able to increase the reliability while using only 2 stereo images instead of 9. With fewer images and several optimizations, we reduced the run time from 15 minutes to less than a minute per step. We also explored using parallel hardware for further speedups.

Section 2 of this paper discusses the constraints used and their effects on system precision. Section 3 presents optimizations for speed and prospects for parallelism. Finally, section 4 presents some extrapolations on the FIDO experience.

The FIDO system has supported experiments in other aspects of visual navigation, notably *interest operators*, used to pick points to be tracked from image to image, and *path planning*. The results have been presented elsewhere [8, 9]. We found that the simple interest operator used in the original Cart program worked as well as more expensive ones, and it was retained with only slight changes. FIDO does incorporate a new, more flexible, path planner based on a grid combinatorial search and incremental path smoothing.

1.1 Constraints

FIDO uses a variety of constraints to improve the accuracy of its stereo vision and motion solutions. Most reduce the area of the image to be searched by the correlator. A smaller search window reduces the chance of finding a false match and improves system performance in several ways. First, as more points are tracked correctly it becomes easier to identify those incorrectly tracked and delete them. Secondly, more points (and higher precision) improve the accuracy of the motion calculations [10]. Finally, points can be successfully tracked through more images, and over longer distances, for more accurate long term navigation.

Some of the constraints arise from the known relationship between the cameras and the vehicle. Other constraints come from vehicle motion estimates: the image location of an object that has been stereo ranged on a previous step is constrained by approximate knowledge of the vehicle's new position.

We tested FIDO using various combinations of constraints in order to judge their effect. We usually made a live vehicle run with the current best settings, and saved all the images and position predictions in a file. Subsequent runs were done off-line using this stored data, with different constraint settings. Such runs were compared for accuracy of the final calculated position, number of features successfully tracked at each step, and occurrence of any catastrophic failures.

1.2 Imaging Geometry Constraints

These constraints are the simplest to understand and to apply. They depend only on camera and robot geometry, and they are applicable to stereo point matches of both new and previously ranged points.

Near and Far Limits. Point distances are not permitted to be greater than infinity (by the real world) or less than a certain distance (by the nose of the robot). This determines a maximum and minimum stereo disparity of the feature match.

Epipolar Constraint. This is the standard stereo epipolar constraint: if the point of view moves purely sideways the image of a point will also move sideways (in the opposite direction) but not up or down. In the real world of misaligned cameras and distorted vidicons, the image might appear to move a little vertically, so we allow some slop (10% of the image height typical).

1.3 Motion Geometry

The estimated motion of the vehicle from step to step places a strong constraint on point matches. It can be used either *a priori* to limit the search area within an image, or *a posteriori* to gauge the reasonableness of a match. The predicted position of the vehicle can also be combined with the points tracked by vision in the vehicle motion calculation. FIDO uses the motion geometry constraints in the following 4 ways:

CH2152-7/85/0000/0830$01.00 © 1985 IEEE

Two D Motion. We usually run our robot on locally flat ground, in which case we know it will not pitch, roll, or move vertically. This reduces the problem of determining vehicle motion from 6 degrees of freedom to 3, simplifying the computation and tightening the constraints.

Reacquire Constraint. Given the 3D location of a point relative to a previous vehicle position, and a dead reckoned new position and heading for the vehicle, it is possible to predict where that point should appear in the new stereo pair of images. If this constraint is active FIDO will use the prediction to limit the stereo matcher's search. Three user-settable variables control the error estimates in robot position and orientation, and consequently the size of the search box around the predicted image position.

Prune. When all points from a previous position have been reacquired at a new vehicle location and stereo-ranged, there is a pruning step that looks for points that do not move rigidly with the rest of the points. The points that do not appear to move rigidly have probably been tracked incorrectly, and can be deleted before the least-squares process that solves for vehicle motion. Activating the Prune constraint causes the predicted vehicle position to be included as one of the points in the rigidity test, perhaps weighting the selection to the correctly matched points rather than a coincidentally consistent incorrect set.

Motion Solution. The motion solver determines the motion that minimizes the error between where points have been seen and where they should have been seen given that motion. The predicted vehicle position can be included as one of the points in this least-squares process, weighted more or less depending on the assumed precision of the prediction.

1.4 Results

We made several runs of the FIDO system on Neptune, with fairly consistent results. Data from June 24, 1984 was most extensively analyzed. On that run a single large obstacle was placed a close 2 meters ahead of Neptune's cameras, with the destination set to the far side. It was a tough test for FIDO, since it required the maximum allowed turn (limited by the need to have significant overlap in the views from successive positions) on each step to get around the obstacle and back on course. We ran FIDO with each constraint in what we thought to be its best state, and saved images and dead reckoning information. Then we made a series of off-line runs on the stored data, varying settings and watching the results. Several runs differed in only one parameter from the original, a few others changed two or three. The last group of runs began with one using none of the constraints, followed by a series each with only one constraint on.

Figure 6 summarizes the results. The most important measure of a run's success is the (program's) calculated position at the end of the run: the nearer to the actual (manually) measured position, the better.

Some cautionary notes are in order. The relative success of the run with only the far distance constraint is accidental. During that run, there were two steps where the motion solution was completely wrong but that by coincidence nearly offset each other. Many of the other single constraint runs that appear worse actually had only one wild miscalculation.

Some of the all-but-one constraint runs also appear too good. In many of these cases the dead-reckoning information was sometimes better than the visual tracking. The run with no epipolar constraint has a better final position than the run with no reacquire constraint, because, by luck, it tracked fewer points at the right times and relied on dead reckoning while the latter placed too much reliance on small numbers of tracked points.

Based on our experiences, we make the following observations:

- The epipolar constraint is the single most powerful constraint. Turning it off, and all the others on, significantly decreases the minimum and average number of features tracked and the accuracy of the motion solution. Turning it on, with all others off, significantly increased the number of points tracked. In a sense, this is not surprising, since the epipolar constraint rules out 90% of the image, more than any other constraint.

- No single constraint makes the difference between a successful and a catastrophic outcome.

- In none of the runs was vision as accurate at calculating translation as straight dead reckoning based on motor commands, though in the best runs vision determined the rotation more correctly. It would have been better to use the dead reckoned motion rather than the visually determined one if the number of features tracked dropped below 6 or 7, rather than 4 which was the threshold, at least for the level of ground roughness and mechanical accuracy in the experiments.

- We noticed that even the best runs have about a 20% error in calculated translation, always on the short side. We suspect a small camera calibration error, and possibly systematic errors in representing uncertainty. FIDO calculated a point's 3D location by projecting rays through the centers of the pixels in the stereo images, which gives a location on the near side of the range of uncertainty of distance.

- There is a problem in using all the geometric constraints to cut down the search area since it leaves none for verification and pruning. If we had very accurate motion prediction, we would have to resort to photometry instead of geometry to identify points that had been occluded or otherwise lost.

2. Speed-up Methods

FIDO now takes 30 to 40 seconds per step on a Vax 11/780 under Unix. To run in real time, we would have to reduce that to about 1 second per step. We have looked at several speed-up techniques, including faster processors, dedicated hardware, coding hacks, and parallel processing.

Faster General Purpose Computers

Our VAX is about a one-MIP (Million Instructions Per second) machine. It is technically possible to get the required speedup by simply obtaining a 30-MIP or faster computer. Budget and logistics leave this as a tantalizing future possibility.

Commercial Array Processors

Buying a commercial array processor is more feasible for us than buying a faster computer. About 90 percent of the runtime in FIDO occurs in image array operations and geometric calculations, particularly the convolutions in point matching. These are done by small pieces of code that work on large amounts of data, and are well suited to the pipelined vector arithmetic of available array

processors. We estimate, for instance, that a 100 MIP array processor could give us the desired factor of 30 speedup. We've made several serious attempts to acquire one; so far, this remains another tantalizing possibility.

Coding optimizations

Much effort has been expended on speeding up the Vax implementation. We feel there is little room for left for significant improvements in a time-shared, paged-memory environment. The basic routines, such as the correlator and the interest operator, fit all the criteria for good candidates for optimization [2]: the code is fairly well understood, stable, small, and accounts for a large amount of run time. For instance, the implementation of the correlator uses the following coding techniques:

- The calculations of parameters of the correlating window are done once, outside the main loop.
- Sums and sums of squares for consecutive columns and rows are calculated by Price's technique [7]. The next window total is calculated by adding in the total for the column that just entered the window and subtracting off the total for the column that just left the window.
- Squares are calculated by table lookup. Since the squares are of sums of two pixel values, the table needs only 511 entries.
- Image windows are moved by pointer swapping, rather than by data transfers.
- Loop indices count down to 0, since the VAX hardware has an efficient test-for-not-0-and-branch instruction.
- Formulas are rewritten to eliminate extra calculations. For example,
  $2 * \Sigma(img1 * img2) =$
  $\Sigma((img1 + img2)\uparrow 2) - \Sigma(img1\uparrow 2) - \Sigma(img2\uparrow 2)$
  gives a way of calculating the sum of the products of the pixel values by additions (which are cheap) and squares (which can be done by table lookup) rather than multiplications. The individual sums are also used in other parts of the calculation, so in this case the sum of products comes for free.
- Loop unrolling. The code in the innermost loop is written n times in line, rather than written once inside a loop that counts to n. This saves n increments of the counter and n tests for the end of the loop.
- Register use. The most frequently used variables are located in hardware registers.

These programming techniques reduce the run time of the correlator from 140 ms per call for a straightforward implementation to 4 to 5 ms per call. Similar optimizations have been performed on the other tight loops, such as in the interest operator and the image fine to coarse reduction routine. The user-level routines have been optimized to the point that the single routine that uses the most CPU time is now an image unpacker.

Dedicated hardware

A dedicated microcomputer running FIDO with enough memory to store all the relevant images offered some hope. We tried an implementation of the correlator on a 10-MHz MC68000 system, with all the images held in integer arrays. After eliminating all floating point operations the resulting code still took 29 microseconds per call to the correlator, compared with 4 to 5 on the VAX.

2.1 Parallelism

There are several ways to break FIDO into separate processes that can run in parallel on different machines, including pipelining on macro or micro scales or the use of a master/slave system.

Macro Pipelining

One process might do the reductions, the next could do reacquires, the next the match, another motion-solving, and the last path planning. This organization improves throughput but not the latency. The problem with this method is the sequential nature of FIDO. Since all the image reductions have to be finished before the reacquires can start, all the matches done before the path planning, and so forth, each pipeline stage has to wait for the previous stage. Since each step takes as long as on a serial machine, and since the steps are done sequentially, the time to process any one set of images is the same as on a single processor system.

Micro Pipelining

The processes could be subdivided more finely. For instance, one processor might do the first level of match for one point after another, handing its results to the process that does the next level of match. When matches are finished, the pipeline could be reconfigured for path planning, and so on. This approach requires huge communication bandwidth between processes.

Master/slave

This method has one master process and several identical slave processes. Each slave handles every image processing task: reduction, matching, and interest operator. At any time all the slaves work on the same task with different data. For example, during image reduction, each slave reduces part of the image, and during matching each slave processes its own queue of points. The master process does tasks that require global knowledge such as path-planning or motion-solving, and coordinates the slaves. This more flexible organization avoids several delays inherent in pipelines.

We implemented variants of this idea in our Ethernet-connected multi-Vax environment. Given the existing uniprocessor code, the task was not difficult. The slaves required new code for communication with the master, but the actual work is done by calls to the old image processing routines. The master contains the old path planning and display code, and new communication code and dispatch tables to keep track of each slave's activities. When a slave completes a task the master updates its dispatch table, finds a new task and puts the slave to work again. For instance during point matching each slave is initially given one point to correlate. When a slave finishes its correlation, the master hands it a new point to find. When all the points are handed out the master redundantly hands out points that are still in process on other slaves, and accepts the first answer to be returned, giving some protection against overloaded or crashed processors.

A version of the system that used several VAXes in parallel was swamped, as expected, by the overhead of squeezing images between machines through the Ethernet. Another version that used multiple processes on a single Vax gave us some idea of the performance that might be possible if faster communication, perhaps through shared memory, were available.

The single machine version uses the same decomposition as the multiple machine version, and the same general-purpose interprocess communication package. Because of limitations in the communications package, each slave calculated its own image pyramid.

2.2 Timings for a 28-Step Run

Single Processor	978	
One Slave		
Master	216	
Slave 1		626
Five Slaves		
Master	234	
Slave 1		403
Slave 2		402
Slave 3		403
Slave 4		402
Slave 5		400

Notes:

- The time for the Master varies little with the number of slaves.
- Without image acquisition or communication package overhead the time for a single slave would be about 325 seconds or 12 seconds per step.
- Without image or communication overhead, and with the time for picture reduction shared evenly, the time for each of the five slaves would be 65 seconds, or about 2.5 seconds per step.
- The work spreads very evenly among the slaves. With 5 slaves, the workload is balanced to within the accuracy of our measurements.
- If the master process did not handle images, had zero-cost communication, and didn't have to do image distortion correction, it could run in 75 to 80 seconds, or about 3 seconds per step.
- By comparison, the original uniprocessor system runs in 978 seconds, or 35 seconds per step. With the advantages we assumed above (no image handling overhead) it would still have taken 503 seconds, or 18 seconds per step.

2.3 Remarks

Our experiments suggest that it is possible to decompose FIDO into a 5 to 10 fold parallel set of efficiently cooperating parts running on conventional processors. To realize the run times suggested above we would need the following:

- Shared main memory large enough to hold at least two image pyramids without swapping or data packing. (2 * [256 + 64 + 16 + 4 + 1 + .25] = 700 KiloBytes).
- Fast interprocess communication for small messages.
- At least 5 processors. It takes 5 slave processors to bring the image processing time into the same range as the master process' time.
- A device able to digitize images directly into the shared memory.
- Cameras with less image distortion than our current vidicons, so image warping would not be needed.

3. The Next System

Some simple hardware enhancements could improve FIDO's performance. A pan mechanism for the stereo cameras would permit larger turns while still maintaining continuity of field of view. Motion and heading sensors would improve navigational accuracy and eliminate some catastrophic misperceptions.

Navigational accuracy could also by improved by modifying the motion estimation algorithm. The current algorithm reacquires features in new a image by searching for the features within windows predicted by an *a priori* motion estimate. This makes poor use of the assumption that objects do not move; that is, that they appear to move rigidly from frame to frame. Since all search windows are defined before any search begins, constraint is not propagated from one match to another. A seemly better approach is the iterative registration method [1], [3], [4]. In this method, 3-D feature positions are projected onto a new image using an initial motion estimate, then the motion estimate is refined to optimize some measure of match in the image. We are currently experimenting with the variation proposed by Lucas [4] and plan to report empirical results in the near future.

Two bugbears in our systems to date have been the calibration of camera and motor parameters and the represention of uncertainty in the 3-D locations of perceived objects. We are considering an adaptive approach that calibrates the cameras (semi-)continuously on the fly and adjusts the motor control parameters from observations of past vehicle motions. A simple technique like this was used successfully in an early program that drove the Stanford Cart in straight lines [5]. We are also looking at carrying along uncertainties in feature locations and updating the uncertainty as new measurements are taken. Eventually, we hope to automate the process to the point where calibration simply requires turning on the vehicle and letting it run by itself for a while.

Acknowledgement

This work has been supported by the Office of Naval Research under contract number N00014-81-K-0503.

Path Relaxation: Path Planning for a Mobile Robot

Charles E. Thorpe

Computer Science Department, Carnegie-Mellon University

Abstract. Path Relaxation is a method of planning safe paths around obstacles for mobile robots. It works in two steps: a global grid search that finds a rough path, followed by a local relaxation step that adjusts each node on the path to lower the overall path cost. The representation used by Path Relaxation allows an explicit tradeoff among length of path, clearance away from obstacles, and distance traveled through unmapped areas.

1. Introduction

Path Relaxation is a two-step path-planning process for mobile robots. It finds a safe path for a robot to traverse a field of obstacles and arrive at its destination. The first step of path relaxation finds a preliminary path on an eight-connected grid of points. The second step adjusts, or "relaxes", the position of each preliminary path point to improve the path.

One advantage of path relaxation is that it allows many different factors to be considered in choosing a path. Typical path planning algorithms evaluate the cost of alternative paths solely on the basis of path length. The cost function used by Path Relaxation, in contrast, also includes how close the path comes to objects (the further away, the lower the cost) and penalties for traveling through areas out of the field of view. The effect is to produce paths that neither clip the corners of obstacles nor make wide deviations around isolated objects, and that prefer to stay in mapped terrain unless a path through unmapped regions is substantially shorter. Other factors, such as sharpness of corners or visibility of landmarks, could also be added for a particular robot or mission.

Path Relaxation is part of Fido, the vision and navigation system of the CMU Rover mobile robot. [7] The Rover, under Fido's control, navigates solely by stereo vision. It picks about 40 points to track, finds them in a pair of stereo images, and calculates their 3D positions relative to the Rover. The Rover then moves about half a meter, takes a new pair of pictures, finds the 40 tracked points in each of the new pictures and recalculates their positions. The apparent change in position of those points gives the actual change in the robot's position.

Fido's world model is not suitable for most existing path-planning algorithms. They usually assume a completely known world model, with planar-faced objects. Fido's world model, on the other hand, contains only the 40 points it is tracking. For each point, the model records its position, the uncertainty in that position, and the appearance of a small patch of the image around that point. Furthermore, Fido only knows about what it has seen; points that have never been within its field of view are not listed in the world model. Also, the vision system may fail to track points correctly, so there may be phantom objects in the world model that have been seen once but are no longer being tracked. All this indicates the need for a data structure that can represent uncertainty and inaccuracy, and for algorithms that can use such data.

Section 2 of this paper outlines the constraints available to Fido's path planner. Section 3 discusses some common types of path planners, and shows how they are inadequate for our application. The Path Relaxation algorithm is explained in detail in Section 4, and some additions to the basic scheme are presented in Section 5. Finally, Section 6 discusses shortcomings of Path Relaxation and some possible extensions.

2. Constraints

An intelligent path planner needs to bring lots of information to bear on the problem. This section discusses some of the information useful for mobile robot path planning, and shows how the constraints for mobile robot paths differ from those for manipulator trajectories.

Low dimensionality. A ground-based robot vehicle is constrained to three degrees of freedom: x and y position and orientation. In particular, the CMU Rover has a circular cross-section, so for path planning the orientation does not matter. This makes path planning only a 2D problem, as compared to a 6 dimensional problem for a typical manipulator.

Imprecise control. Even under the best of circumstances, a mobile robot is not likely to be very accurate: perhaps a few inches, compared to a few thousandths of an inch for manipulators. The implication for path planning is that it is much less important to worry about exact fits for mobile robot paths. If the robot could, theoretically, just barely fit through a certain opening, then in practice that's probably not a good way to go. Computational resources are better spent exploring alternate paths rather than worrying about highly accurate motion calculations.

Cumulative error. Errors in a dead-reckoning system tend to accumulate: a small error in heading, for instance, can give rise to a large error in position as the vehicle moves. The only way to reduce error is to periodically measure position against some global standard, which can be time-consuming. The Rover, for example, does its measurement by stereo vision, taking a few minutes to compute its exact position. So a slightly longer path that stays farther away from obstacles, and allows longer motion between stops for measurement, may take less time to travel than a shorter path that requires more frequent stops. In contrast, a manipulator can reach a location with approximately the same error regardless of what path is taken to arrive there. There is no cumulative error, and no time spent in reorientation.

Unknown areas. Robot manipulator trajectory planners usually know about all the obstacles. The Rover knows only about those that it has seen. This leaves unknown areas outside its field of view and behind obstacles. It is usually preferable to plan a path that traverses only known empty regions, but if that path is much longer than the shortest path it may be worth looking at the unknown regions.

Fuzzy objects. Not only do typical manipulator path-planners know about all the objects, they know precisely where each object is. This information might come, for instance, from the CAD system that designed the robot workstation. Mobile robots, on the other hand, usually sense the world as they go. Fido, instead of having precise

bounds for objects, knows only about fuzzy points. The location of a point is only known to the precision of the stereo vision system, and the extent of an object beyond the point is entirely unknown.

In summary, a good system for mobile robot path planning will be quite different from a manipulator path planner. Mobile robot path planners need to handle uncertainty in the sensed world model and errors in path execution. They do not have to worry about high dimensionality or extremely high accuracy. Section 3 of this paper discusses some existing path planning algorithms and their shortcomings. Section 4 then presents the algorithms used by Path Relaxation, and shows how they address these problems.

3. Approaches to Path Planning

This section outlines several approaches to path planning and some of the drawbacks of each approach. All of these methods except the potential fields approach abstract the search space to a graph of possible paths. This graph is then searched by some standard search technique, such as breadth-first or A* [8], and the shortest path is returned. The important thing to note in the following is the information made explicit by each representation and the information thrown away.

Free Space methods. [2, 3, 9] One type of path planner explicitly deals with the space between obstacles. Paths are forced to run down the middle of the corridors between obstacles, for instance on the Voronoi diagram of the free space. Free space algorithms suffer from two related problems, both resulting from a data abstraction that throws away too much information. The first problem is that paths always run down the middle of corridors. In a narrow space, this is desirable, since it allows the maximum possible robot error without hitting an object. But in some cases paths may go much further out of their way than necessary. The second problem is that the algorithms do not use clearance information. The shortest path is always selected, even if it involves much closer tolerances than a slightly longer path.

Vertex Graphs. [5, 10, 6] Another class of algorithms is based on a graph connecting pairs of vertices. For each pair of vertices, if the line between them does not intersect any obstacle, that line is added to the graph of possible paths. Vertex graph algorithms suffer from the "too close" problem: in their concern for the shortest possible path, they find paths that clip the corners of obstacles and even run along the edges of some objects. It is, of course, possible to build in a margin of error by growing the obstacles by an extra amount; this may, however, block some paths.

Both free space and vertex graph methods throw away too much information too soon. All obstacles are modeled as polygons, all paths are considered either open or blocked, and the shortest path is always best. There is no mechanism for trading a slightly longer path for more clearance, or for making local path adjustments. There is also no clean way to deal with unmapped regions, other than to close them off entirely.

The **Potential Fields** [1, 4] approach tries to make those tradeoffs explicit. Conceptually, it turns the robot into a marble, tilts the floor towards the goal, and watches to see which way the marble rolls. Obstacles are represented as hills with sloping sides, so the marble will roll a prudent distance away from them but not too far, and will seek the passes between adjacent hills. The problem with potential field paths is that they can get caught in dead ends: once the marble rolls into a box canyon, the algorithm has to invoke special-case mechanisms to cut off that route, backtrack, and start again. Moreover, the path with the lowest threshold might turn out to be a long and winding road, while a path that must climb a small ridge at the start and then has an easy run to the goal might never be investigated.

Another approach that could explicitly represent the conflicts between short paths and obstacle avoidance is the **Regular Grid** method. This covers the world with a regular grid of points, each connected with its 4 or 8 neighbors to form a graph. In existing regular grid implementations, the only information stored at a node is whether it is inside an object or not. Then the graph is searched, and the shortest grid path returned. This straightforward grid search has many of the same "too close" problems as the vertex graph approaches.

4. Path Relaxation

Path Relaxation combines the best features of grid search and potential fields. Using the rolling marble analogy, the first step is a global grid search that finds a good valley for the path to follow. The second step is a local relaxation step, similar to the potential field approach, that moves the nodes in the path to the bottom of the valley in which they lie. The terrain (cost function) consists of a gradual slope towards the goal, hills with sloping sides for obstacles, and plateaus for unexplored regions. The height of the hills has to do with the confidence that there really is an object there. Hill diameter depends on robot precision: a more precise robot can drive closer to an object, so the hills will be tall and narrow, while a less accurate vehicle will need more clearance, requiring wide, gradually tapering hillsides.

This section first presents results on how large the grid size can be without missing paths. It next discusses the mechanism for assigning cost to the nodes and searching the grid. Finally, it presents the relaxation step that adjusts the positions of path nodes.

Grid Size. How large can a grid be and still not miss any possible paths? That depends on the number of dimensions of the problem, on the connectivity of the grid, and on the size of the vehicle. It also depends on the vehicle's shape: in this section, we discuss the simplest case, which is a vehicle with a circular cross-section.

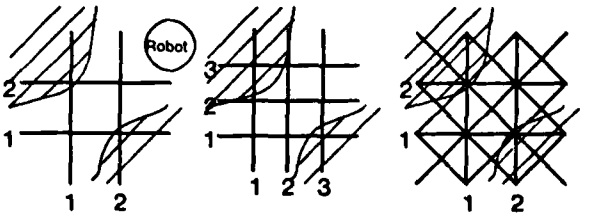


Figure 1: Grid Size Problems

The area to be traversed can be covered with a grid in which each node is connected to either its four or its eight nearest neighbors. For a four-connected grid, if the spacing were r, there would be a chance of missing diagonal paths. At left in Figure 1, for instance, there is enough room for the robot to move from (1,1) to (2,2), yet both nodes (1,2) and node (2,1) are blocked. To guarantee that no paths are missed, the grid spacing must be reduced to $r * sqrt(2) / 2$, as in the center of Figure 1. That is the largest size allowable that guarantees that if diagonally opposite nodes are covered, there is not enough room between them for the robot to safely pass. Note that the converse is not necessarily true: just because there is a clear grid path does not guarantee that the robot will fit. At this stage, the important thing is to find all possible paths, rather than to find only possible paths.

If the grid is eight-connected, as in the right of Figure 1, (each node connected to its diagonal, as well as orthogonal, neighbors), the problem with diagonal paths disappears. The grid spacing can be a full r, while guaranteeing that if there is a path it will be found.

Grid Search. Once the grid size has been fixed, the next step is to assign costs to paths on the grid and then to search for the best path along the grid from the start to the goal. "Best", in this case, has three conflicting requirements: shorter path length, greater margin away from obstacles, and less distance in uncharted areas. These three are explicitly balanced by the way path costs are calculated. A path's cost is the sum of the costs of the nodes through which it passes, each multiplied by the distance to the adjacent nodes. (In a 4-connected graph all lengths are the same, but in an 8-connected graph we have to distinguish between orthogonal and diagonal links.) The node costs consist of three parts to explicitly represent the three conflicting criteria.

1. Cost for distance. Each node starts out with a cost of one unit, for length traveled.

2. Cost for near objects. Each object near a node adds to that node's cost. The nearer the obstacle, the more cost it adds. The exact slope of the cost function will depend on the accuracy of the vehicle (a more accurate vehicle can afford to come closer to objects), and the vehicle's speed (a faster vehicle can afford to go farther out of its way), among other factors.

3. Cost for within or near an unmapped region. The cost for traveling in an unmapped region will depend on the vehicle's mission. If this is primarily an exploration trip, for example, the cost might be relatively low. There is also a cost added for being near an unmapped region, using the same sort of function of distance as is used for obstacles. This provides a buffer to keep paths from coming too close to potentially unmapped hazards.

The first step of Path Relaxation is to set up the grid and read in the list of obstacles and the vehicle's current position and field of view. The system can then calculate the cost at each node, based on the distances to nearby obstacles and whether that node is within the field of view. The next step is to create links from each node to its 8 neighbors. The start and goal locations do not necessarily lie on grid points, so special nodes need to be created for them and linked into the graph. Links that pass through an obstacle, or between two obstacles with too little clearance for the vehicle, can be detected and deleted at this stage.

The system then searches this graph for the minimum-cost path from the start to the goal. The search itself is a standard A* [8] search. The estimated total cost of a path, used by A* to pick which node to expand next, is the sum of the cost so far plus the straight-line distance from the current location to the goal. This has the effect, in regions of equal cost, of finding the path that most closely approximates the straight-line path to the goal.

The path found is guaranteed to be the lowest-cost path on the grid, but this is not necessarily the overall optimal path. First of all, even in areas with no obstacles the grid path may be longer than a straight-line path simply because it has to follow grid lines. For a 4-connected grid, the worst case is diagonal lines, where the grid path is sqrt(2) times as long as the straight-line path. For an 8-connected grid, the equivalent worst case is a path that goes equal distances forward and diagonally. This gives a path about 1.08 times as long as the straight-line path. In cases where the path curves around several obstacles, the extra path length can be even more significant. Secondly, if the grid path goes between two obstacles, it may be non-optimal because a node is placed closer to one obstacle than to the other. A node placed exactly half way between the two obstacles would, for most types of cost functions, have a lower cost. The placement of the node that minimizes the overall path cost will depend both on node cost and on path length, but in any case is unlikely to be exactly on a grid point. If the grid path is topologically equivalent to the optimal path (i.e. goes on the same side of each object), the grid path can be iteratively improved to approximate the optimal path (see Section 5). But if the grid path at any point goes on the "wrong" side of an obstacle, then no amount of local adjustment will yield the optimal path. The chance of going on the wrong side of an obstacle is related to the size of the grid and the shape of the cost vs. distance function. For a given grid size and cost function, it is possible to put a limit on how much worse the path found could possibly be than the optimal path. If the result is too imprecise, the grid size can be decreased until the additional computation time is no longer worth the improved path.

A few details on the shape of the cost function deserve mention. Many different cost functions will work, but some shapes are harder to handle properly. The first shape we tried was linear. This had the advantage of being easy to calculate quickly, but gave problems when two objects were close together. The sum of the costs from two nearby objects was equal to a linear function of the sum of the distances to the objects. This creates ellipses of equal cost, including the degenerate ellipse on the line between the two objects. In that case, there was no reason for the path to pick a spot midway between the objects, as we had (incorrectly) expected. Instead, the only change in cost came from changing distance, so the path went wherever it had to to minimize path length. In our first attempt to remedy the situation we replaced the linear slope with an exponentially decaying value. This had the desired effect of creating a saddle between the two peaks, and forcing the path towards the midpoint between the objects. The problem with exponentials is that they never reach zero. For a linear function, there was a quick test to see if a given object was close enough to a given point to have any influence. If it was too far away, the function did not have to be evaluated. For the exponential cost function, on the other hand, the cost function had to be calculated for every obstacle for each point. We tried cutting off the size of the exponential, but this left a small ridge at the extremum of the function, and paths tended to run in nice circular arcs along those ridges. A good compromise, and the function we finally settled on, is a cubic function that ranges from 0 at some maximum distance, set by the user, to the obstacle's maximum cost at 0 distance. This has both the advantages of having a good saddle between neighboring obstacles and of being easy to compute and bounded in a local area.

Relaxation. Grid search finds an approximate path; the next step is an optimization step that fine-tunes the location of each node on the path to minimize the total cost. One way to do this would be to precisely define the cost of the path by a set of non-linear equations and solve them simultaneously to analytically determine the optimal position of each node. This approach is not, in general, computationally feasible. The approach used here is a relaxation method. Each node's position is adjusted in turn, using only local information to minimize the cost of the path sections on either side of that node. Since moving one node may affect the cost of its neighbors, the entire procedure is repeated until no node moves farther than some small amount.

Node motion has to be restricted. If nodes were allowed to move in any direction, they would all end up at low cost points, with many nodes bunched together and a few long links between them. This would not give a very good picture of the actual cost along the path. So in order to keep the nodes spread out, a node's motion is restricted to be perpendicular to a line between the preceding and following nodes. Furthermore, at any one step a node is allowed to move no more than one unit.

As a node moves, all three factors of cost are affected: distance traveled (from the preceding node, via this node, to the next node), proximity to objects, and relationship to unmapped regions. The combination of these factors makes it difficult to directly solve for minimum cost node

position. Instead, a binary search is used to find that position to whatever accuracy is desired.

The relaxation step has the effect of turning jagged lines into straight ones where possible, of finding the "saddle" in the cost function between two objects, and of curving around isolated objects. It also does the "right thing" at region boundaries. The least cost path crossing a border between different cost regions will follow the same path as a ray of light refracting at a boundary between media with different transmission velocities. The relaxed path will approach that path.

5. Additions to the Basic Scheme

One extension we have tried is to vary the costs of individual obstacles. The current vision system sometimes reports phantom objects, and sometimes loses real objects that it had been tracking correctly. The solution to this is to "fade" objects by decreasing their cost each step that they are within the field of view but not tracked by the vision module.

Another extension implemented is to re-use existing paths whenever possible. At any one step, the vehicle will usually move only a fraction of the length of the planned path. If no new objects are seen during that step, and if the vehicle is not too far off its planned course, it is possible to use the rest of the path as planned. Only if new objects have been seen that block the planned path is it necessary to replan from scratch.

The relaxation step can be greatly speeded up if it runs in parallel on several computers. Although an actual parallel implementation has not yet been done, a simulation has been written and tested.

6. Remaining Work

Path Relaxation would be easy to extend to higher dimensions. It could be used, for example, for a 3D search to be used by underwater vehicles maneuvering through a drilling platform. Another use for higher-dimensional searches would be to include rotations for asymmetric vehicles. Yet another application would be to model moving obstacles; then the third dimension becomes time, with the cost of a grid point having to do with distance to all objects at that time. This would have a slightly different flavor than the other higher-dimensional extensions; it is possible to go both directions in x, y, z, and theta, but only one direction in the time dimension.

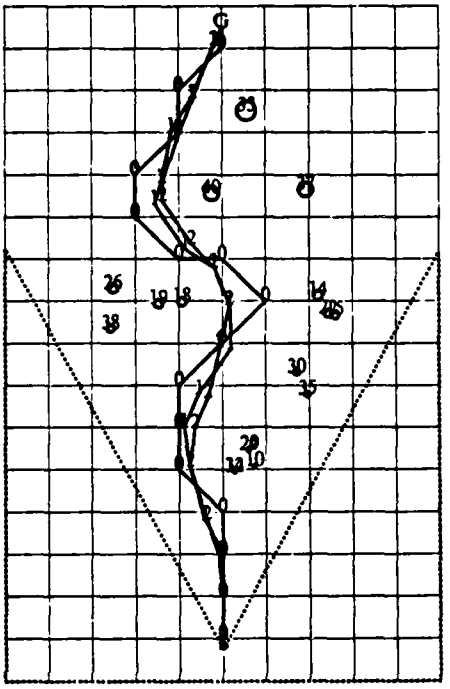
Another possible extension has to do with smoothing out sharp corners. All wheels on the Rover steer, so it can follow a path with sharp corners if necessary. Many other vehicles, are not so maneuverable; they may steer like a car, with a minimum possible turning radius. In order to accommodate those vehicles, it would be necessary to restrict both the graph search and relaxation steps. A related problem is to use a smoothly curved path rather than a series of linear segments.

An interesting direction to pursue is multiple-precision grids. This could make it possible to spend more effort working on precise motion through cluttered areas, and less time on wide open spaces.

Path relaxation, as well as almost all existing path planners, deals only with geometric information. A large part of a robot's world knowledge, however, may be in partially symbolic form. For example, a map assembled by the vehicle itself may have very precise local patches, each measured from one robot location. The relations between patches, though, will probably be much less precise, since they depend on robot motion from one step to the next. Using such a mixture of constraints is a hard problem.

Acknowledgements Thanks to Hans Moravec, Larry Matthies, and Rich Wallace for advice and encouragement. This research was partially supported by Office of Naval Research contract N00014-81-K-0503.

Example Run. Figure 2 is a run from scratch, using real data extracted from images by the Fido vision system. The circles are obstacles, where the size of the circle is the uncertainty of the stereo vision system. The dotted line surrounds the area out of the field of view. The start position of the robot is approximately (0, -.2) and the goal is (0, 14.5). The grid path found is marked by 0's. After one iteration of relaxation, the path is marked by 1's, and after the second (and, in this case, last) relaxation, by 2's.



References

1. J. Randolph Andrews. Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator. Master Th., MIT, 1983.

2. Rodney Brooks. Solving the Find-Path Problem by Representing Free Space as Generalized Cones. AI Memo 674, Massachusetts Institute of Technology, May, 1982.

3. Georges Giralt, Ralph Sobek, and Raja Chatila. A Multi-Level Planning and Navigation System for a Mobile Robot; A First Approach to Hilare. Proceedings of IJCAI-6, August, 1979.

4. Oussama Khatib. Dynamic Control of Manipulators in Operational Space. Sixth CISM-IFToMM Congress on Theory of Machines and mechanisms, New Delhi, India, December, 1983.

5. Tomas Lozano-Perez and Michael A. Wesley. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles." *CACM 22*, 10 (October 1979).

6. Hans Moravec. Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. Tech. Rept. CMU-RI-TR-3, Carnegie-Mellon Univesity Robotics Institute, September, 1980.

7. Hans Moravec. The CMU Rover. Proceedings of AAAI-82, August, 1982.

8. N. Nilsson. *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill, 1971.

9. Colm O'Dunlaing, Micha Sharir, and Chee Yap. Retraction: a new approach to motion-planning. Courant Institute, November, 1982.

10. Alan M. Thompson. The Navigation System of the JPL Robot. Proceedings of IJCAI-5, 1977.

Modelling Uncertainty in 3D Stereo Navigation

Larry Matthies
Computer Science Department
Carnegie-Mellon University

Abstract

We are studying the accuracy with which stereo vision can guide a mobile robot. In stereo navigation, a robot uses a sequence of stereo images to estimate its own motion as it travels through a world of stationary objects. A set of landmarks is established by finding corresponding features in one stereo pair. This yields an initial 3-D model of the local environment of the robot, defined in robot-centered coordinates. As the robot moves, it periodically digitizes another stereo pair, finds the landmarks in the new images, and computes their coordinates relative to its new location. The motion of the robot since the last stereo pair is determined by fitting a transformation mapping between the new and the old coordinate values.

Previous algorithms for stereo navigation have suffered from poor accuracy and low tolerance to correspondence errors. This is partly due to inadequate models of stereo triangulation error. Typically, scalar reliability factors are associated with landmarks to indicate the uncertainty in their 3-D coordinates. These scalars are used to weight the contribution of each landmark in the motion solving algorithm. This paper argues that stereo triangulation error is better modelled by treating landmark locations as random variables with 3-D normal distributions. This leads to revised algorithms for motion solving in which the covariance matrices weight the contribution of each landmark. Preliminary simulation results show that the matrix weights achieve substantially more accurate motion estimates than scalar weights. These results should carry over into applications of 3-D vision outside of navigation.

1. Introduction

Mobile robot navigation is a problem of growing interest and practical importance. A travelling robot must be able to detect the shapes and positions of nearby objects and to monitor its own position in a global reference frame. This requires range sensors and motion sensors; we are currently exploring stereo vision for use as both.

Our paradigm for stereo navigation operates as follows [13]. For simplicity, assume that nothing in the environment moves except for the robot. A set of landmarks is defined in a robot-centered coordinate system by matching features in a pair of stereo images. The robot then takes a step, finds

the landmarks in a new pair of images, and calculates their coordinates relative to its new location. The motion between stereo pairs is reflected in the difference between the new and the old landmark coordinates; an estimate of this transformation is found with least squares. The whole process is repeated periodically to monitor robot motion over long distances.

We have previously used this paradigm in systems that were able to guide a robot through short obstacle courses [13], [17]. In one set of experiments, the robot accumulated approximately half a meter of error in its global position estimate over a course six meters long [11]. However, the motion estimates were rather unstable. This instability is reflected throughout the computer vision literature: algorithms for visual motion estimation are generally very sensitive to noisy data [2].

Part of this sensitivity is due to inadequate modelling of stereo triangulation error. Triangulation induces an uncertainty on 3-D coordinates that is greater for distant points than for near points and greater in the direction of the line of sight than perpendicular to it (see figure 2). This phenomenon has been recognized and modelled for a long time in photogrammetry [15], but has been comparatively ignored in computer vision. In photogrammetry it is common to model all measurements as corrupted by normally distributed noise. 3-D positions inferred by triangulation have an uncertainty modelled by 3-D normal distributions. In computer vision, Blostein and Huang [2] have recently derived other probabilistic models of triangulation error, but they appear not to use them in their algorithm for motion solving. Moravec's system [13] approximated triangulation error with scalar coefficients used to weight the contribution of each landmark to the motion solution. However, this does not capture the elongated and oriented nature of the uncertainty.

The purpose of this paper is to demonstrate the importance of modelling triangulation error. The next section shows how 3-D normal distributions modelling the uncertainty in landmark positions can be inferred from stereo data. This model is used in section three to derive new equations for estimating motion. In these equations the covariance matrices of the normal distributions replace the scalar weights of previous methods. Section four shows how to update the local 3-D model with measurements from successive stereo pairs. It proposes to keep the representation in robot-centered coordinates and shows how to use the error model to weight successive range measurements of point locations. Only translational motion is treated. In section five we discuss the cascading of incremental robot motion estimates to obtain an estimate of the global robot position and positional uncertainty. The results of simulations on synthetic data are presented in section six. These compare the new error model with a scalar weighting scheme and show substantially better performance with the new model. Finally, the last section discusses the significance of these results, the difficulties we expect to have in transferring them to real images, and our plans for extending the work.

2. Modelling Stereo Triangulation Error

The geometry of stereo triangulation is shown in figure 1. For the moment we consider just the 2-D case in which two dimensional points project onto one dimensional images. Two cameras are placed at offsets of $\pm b$ from a coordinate system centered between the cameras. Given the coordinates x_l and x_r of the left and right images of the point P, the coordinates of P are given by

$$X_P = \frac{b(x_l + x_r)}{x_l - x_r}$$

$$Y_P = \frac{2b}{x_l - x_r} \tag{1}$$

This estimate can be in error for several reasons. The finite resolution of the images contributes a quantization uncertainty shown in figure 2a. A point projecting to pixels x_l and x_r can lie anywhere in the shaded region. As shown in figure 2b, this region grows with the distance to the point, becomes more skewed with increasing distance, and is always directed along the line of sight to the point. Besides this quantization effect, the stereo matcher can return slightly incorrect values of x_l and x_r due to perspective and photometric distortions of the image. On top of this their may be geometric distortions in the image or calibration errors between the two cameras. These errors are of a more random nature, but they all contribute uncertainty similar to that shown in figure 2.

Our goal is to find a model that accurately reflects the nature of this uncertainty and that can be used conveniently to constrain algorithms for motion solving. Scalar weights can capture the "size" of the uncertainty, but nothing of its shape. In a slightly different context, Baird [1] used polygons to outline the border of the uncertainty region. These became constraints in a motion solving algorithm based on linear programming. In our situation the random nature of the errors makes a statistical approach more appropriate. Motivated largely by the example of photogrammetry and the stereo calibration work of Gennery [7], we model the image coordinates as random variables with known distributions and derive distributions on the point coordinates. For simplicity, we use linear models and normal distributions throughout, rather than try to determine exact distributions from nonlinear functions.

We begin by treating x_l and x_r in equation (1) as corrupted by zero-mean, gaussian (normally distributed) noise; that is,

$$x_l = \mathrm{x}_l + e_l$$
$$x_r = \mathrm{x}_r + e_r$$

where $e_l \sim N(0,\sigma_l)$, $e_r \sim N(0,\sigma_r)$, and x_l and x_r are the true values of x_l and x_r. Since (1) is nonlinear, X_P and Y_P will not be normally distributed. However, we will approximate them as binormal, with means given by (1) and covariances obtained by linearization. Thus,

$$P = \mathrm{P} + e_P \tag{1.5}$$

$$e \sim N(0,V_P)$$
$$\text{and } V_P = JVJ^T$$

Here P is the true value of P, e_P is its random component, J is the Jacobian of (1), and V is the 2×2 covariance matrix of the image coordinates. In the model we have described, V will have σ_l and σ_r on the diagonal and zeroes off the diagonal, since we are assuming there is no correlation between images.

Figure 1.

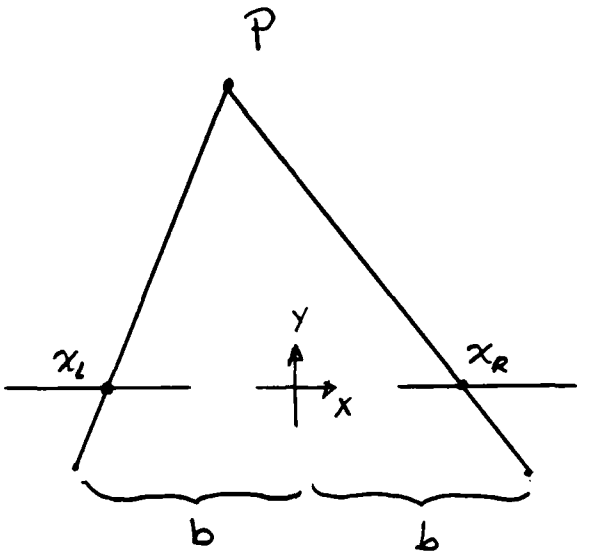


Figure 2a

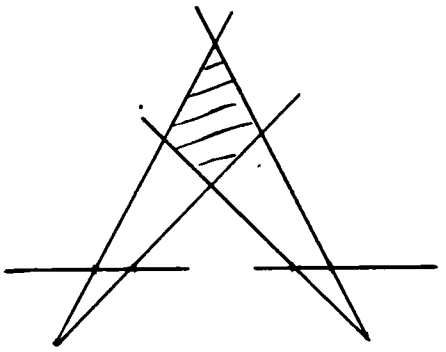
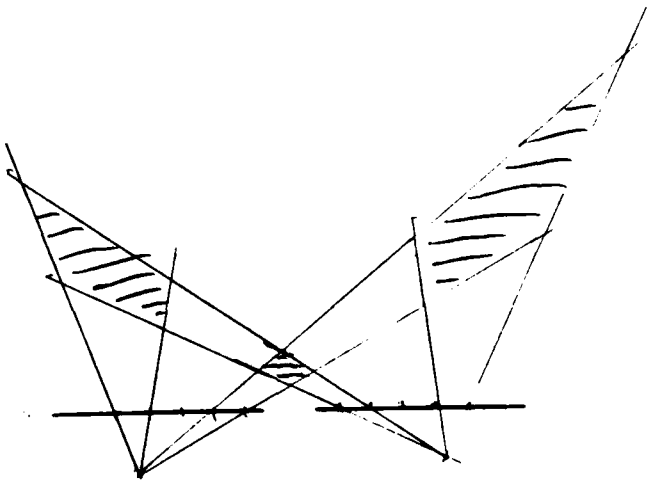


Figure 2b



Note that constant probability contours of the error distributions describe ellipses that approximate the shape of both the non-random (figure 2) and the random contributions to landmark uncertainty. The principal shortcoming of the model we have proposed is that it is not long-tailed as the true distribution would be. Figure 2 hints at this; the uncertainty regions have a skew that isn't modelled by a symmetric distribution. The skew is not significant for nearby points, but grows with distance. We have not analyzed the effect of this other than by way of the simulations presented later. The extension of this error model to 3-D points projecting onto 2-D images is straightforward.

3. Solving for Robot Motion

With the procedure above, 3-D coordinates and covariance matrices are estimated for a number of points matched in the first stereo pair. After the robot moves and digitizes another stereo pair, we find the same features in the new images, triangulate, and compute new covariance matrices. This leads to two models of the same points, with coordinates differing by the motion of the robot. If the robot approached a landmark there will be less measurement error in the landmark coordinates, so the terms of its covariance matrix will be smaller. The opposite will be true if the robot receded from the landmark. See figure 3.

We now wish to determine the motion of the robot between stereo pairs. Suppose for the moment that the motion is purely translational. Let P_i represent landmark coordinates with respect to the first robot position, Q_i represent the coordinates with respect to the second position, and $T = [T_x\ T_y\ T_z]^T$ be the unknown translation vector. The motion is described simply by

$$Q_i = P_i + T \tag{2}$$

In (2) we have observations of P_i and Q_i and wish to find T. The standard method is to apply least squares to minimize

$$\sum_{i=1}^{n} (Q_i - P_i - T)^T (Q_i - P_i - T) \tag{3}$$

Differentiating, setting the result to zero, and solving for T we obtain

$$T = \frac{1}{n} \sum_{i=1}^{n} (Q_i - P_i)$$

When one has information on the reliability of each point, as we do here, the terms in the sum are typically weighted according to their reliability. For scalar weights this modifies expression (3) to be

$$\sum_{i=1}^{n} w_i (Q_i - P_i - T)^T (Q_i - P_i - T) \tag{4}$$

with the resulting motion solution given by

Figure 3

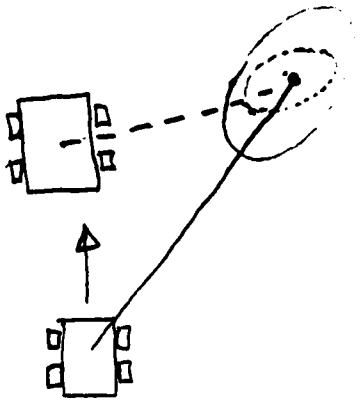
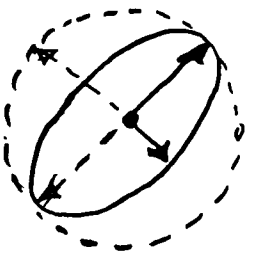
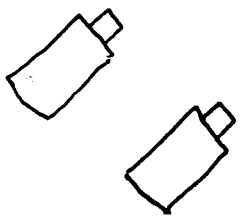


Figure 4





CAMERAS

$$T = \frac{1}{n}\sum_{i=1}^{n}(Q_i - P_i)$$

With the new error model we proceed differently. Since P_i and Q_i are treated as normally distributed vectors, the motion equation (1.5) can be rewritten as

$$T = Q_i - P_i = D_i \tag{5}$$

where D_i will be a normal vector with distribution $N(0, V_{Pi} + V_{Qi}) = N(0, V_i)$. Equation (5) is a linear statistical model whose optimal solution can be reached several different ways [5]. One of these is to minimize the following least squares expression:

$$\sum_{i=1}^{n}(Q_i - P_i - T)^T V_i^{-1}(Q_i - P_i - T) \tag{6}$$

$$= \sum_{i=1}^{n}(Q_i - P_i - T)^T W_i(Q_i - P_i - T)$$

This is equation (4) with the scalar weights w_i replaced by the matrix weights W_i (the inverses of the covariance matrices V_i). The solution for T is

$$T = (\sum_{i=1}^{n} W_i)^{-1} \sum_{i=1}^{n}[W_i(Q_i - P_i)] \tag{7}$$

The inverse covariance matrices in (6) have the effect of replacing the usual Euclidean distance norm, represented by the vector dot product in (4), with new norms for each point that stretch the space as appropriate for the error in that point. This is shown in figure 4. Without the matrix weights, residual vectors lying on circular contours have equal contributions to the total error of fit; with the matrices, these contours become elliptical. This effectively gives more weight to errors perpendicular to the line of sight than parallel to it, which intuitively is what we want. In fact, scalar weights are just the special case of matrices in which the matrix is diagonal with all diagonal elements equal, ie. $W_i = w_i I$.

Since the translation T is given as a linear combination of normal random vectors, it will itself be a normal random vector. The mean of its distribution is simply the value computed by equation (7). The covariance matrix is given by

$$V_T = (\sum_{i=1}^{n} W_i)^{-1}$$

This matrix can be analyzed to determine the quality of the motion estimate.

All of the foregoing was derived assuming that the robot motion was purely translational. This is convenient because the equations remain linear, allowing solutions to be obtained simply and

preserving the normal error model. In the case of general motion, the presence of rotation introduces a nonlinearity that complicates matters. The motion is now expressed by

$$Q_i = R P_i + T \tag{8}$$

where R is a 3×3 rotation matrix. The standard least squares approach would find R and T by minimizing

$$\sum_{i=1}^{n} w_i (Q_i - R P_i - T)^T (Q_i - R P_i - T) \tag{9}$$

Since the matrix R is a complicated function of the rotation angles, the equations obtained by differentiating are nonlinear. The original approach to solving them was to linearize and iterate; however, recently two methods have been found to obtain a solution directly. In first, Hebert [10] expressed the rotation as a quaternion and found a direct solution by applying certain identities in quaternion algebra. The other is a technique from statistics called Procrustes analysis that solves the matrix formulation directly [14]. Both of these methods apply to equations such as (9) that involve only scalar weights, but fail when matrix weights are used. Applying our error model to general motion leads to minimizing

$$\sum_{i=1}^{n} (Q_i - R P_i - T)^T W_i (Q_i - R P_i - T) \tag{10}$$

$$\text{with } W_i = (R V_{Pi} R^T + V_{Qi})^{-1}$$
$$\text{where } P_i \sim N(0, V_{Pi}) \text{ and } Q_i \sim N(0, V_{Qi})$$

The only method we have found for solving this equation is iterative. An initial approximation is obtained using the Procrustes method with scalar weights, then several iterations are performed on a linearized version of (10). Since the initial approximation is close to the solution, weight matrices W_i are calculated only once with the initial approximation for R, rather than recalculated every iteration.

As in the purely translational case, the computed motion parameters are random vectors, but because of the nonlinearity of the rotation they are no longer normally distributed. A normal approximation to the true distribution can be obtained from the converged solution to (10).

4. Updating the Local Model

The foregoing triangulation and motion solving algorithms provide a series of 3-D models defined relative to successive robot locations. Combining these models can serve two purposes. First, averaging landmark sightings from several views should provide more accurate estimates of the landmark positions, which should in turn lead to more accurate estimates of robot motion. Second, all of the models can be incorporated into a single map of the entire area traversed. Previous approaches to these tasks differ according to whether they have an incremental or a batch nature.

One of the best examples of a batch approach is the classical photogrammetric block adjustment

[15]. The problem here is to find the 3-D coordinates of ground points from their correspondences in a block of overlapping aerial photographs. The solution involves writing a set of simultaneous equations relating all of the image coordinates to the unknown ground points and camera positions, then solving for the unknowns via least squares. Typically, all of the measurements and all of the unknowns are treated as normally distributed random variables, much as we have just done. A large aerial survey may involve several hundred unknowns.

The drawbacks of this approach are that it is expensive in time and space, it is difficult to find errors in the mass of data, and its off-line nature makes it inappropriate for continuous, real-time navigation. Photogrammetrists have responded to these problems with an incremental technique called on-line photogrammetry [8]. This method processes new measurements sequentially to update previous estimates of camera and ground point positions, rather than first accumulating all measurements and then estimating the unknowns. Kalman filters are used for the update process. On-line photogrammetry is used as an automation aid when processing aerial images and as an initial screen for erroneous measurements, but it appears that the batch solution is still used to deliver the final values for coordinates.

In computer vision, the best example of an incremental technique is the system developed by Hallam [9]. This involved a 2-D world in which a moving submersible used sonar to track moving and stationary targets. The positions and velocities of the robot and the targets were modelled as state vectors defined in a fixed, global coordinate system. Incoming sonar readings created a local model of the targets in robot-centered coordinates. The current robot parameters were estimated from the difference between the local and global target models, then added to the local target models to update the global target positions and velocities. Kalman filters were the basis for the state updates. Errors in the sonar data were modelled by 2-D normal distributions. This system was found to work quite well on simulated data, but has not yet been applied to real data.

Broida and Chellappa [3] have taken a similar approach to motion estimation from a monocular image sequence. They estimate the position and velocity of a single moving object seen by a stationary camera. Feature correspondences are used as input to a Kalman filter-based state update.

Chatila and Laumond have developed an incremental navigation system for a robot equipped with a laser range finder and an odometer [4]. The robot is modelled as travelling through 2-D world of stationary, polygonal obstacles. The key features of their system are that it uses a scalar model of uncertainty similar to Moravec [13] and that object models are rooted in a common global coordinate frame. Their approach to world model update is intermediate between classical photogrammetry and recursive filtering; when new information on robot position arrives, they percolate this backward to update positions of previously seen objects. This effect "fades", so that the percolation stops after a short time.

In our problem we are concerned with stationary points (landmarks) seen from a moving vehicle. We adopt an update method similar to Hallam, but keep the landmark coordinates in *robot-centered* rather than global coordinates. For example, consider the situation after solving for the first step of

robot motion (figure 5). We have landmark sightings obtained from the previous robot location, sightings from the current robot location, and an estimate of the intervening motion. Covariance matrices are associated with all landmark positions and the robot motion. We propose to transform the previous sightings into the current coordinate frame, average the two sets of coordinates, and use the result as a new, robot-centered estimate of the landmark locations. The transformation and averaging will result in new covariance matrices for the landmarks that should represent diminished uncertainty in their *robot-centered* coordinates.

The rationale for this approach lies in the uncertainty of the motion estimate. For a robot travelling in an unknown environment, its position relative to any fixed reference frame must become more and more uncertain. If new landmark sightings are related back to this fixed frame, then their positions in the fixed frame also become more and more uncertain. Thus, if we transform new measurements back to an old frame for the sake of averaging, we inflate the uncertainty of the new measurements and degrade their contribution to the average. Unfortunately, for a robot travelling forward the most recent stereo measurements will be the most accurate and should be weighted the most heavily; transforming backward will weight it the least heavily. Therefore, in what follows we transform information forward to maintain the landmark coordinates in robot-centered coordinate frame. We expect that this will lead to better estimates of step-by-step robot motion, although other procedures may be preferred for mapping the area covered in several robot steps.

We will treat only translational motion. Let P_i be the robot-centered coordinates of a landmark at time i and P_{i+1} be its updated, robot-centered coordinates at time $i+1$. P_i is transformed to the $i+1^{st}$ coordinate frame by

$$P_i' = P_i + T \tag{11}$$

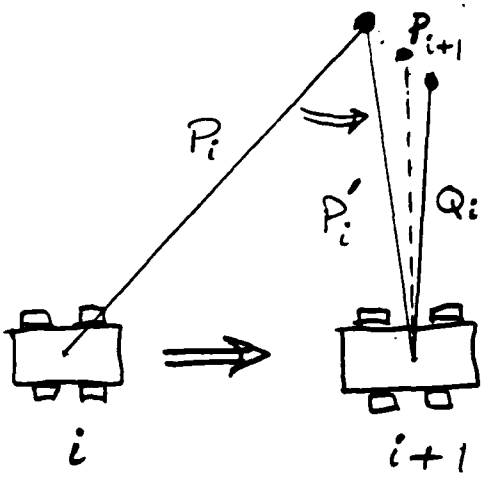
where T is the intervening robot motion. Since we are modelling both P_i and T as corrupted with zero-mean, gaussian noise with known covariance, P_i' will also have a zero-mean, gaussian noise component. If the noise in P_i is $\varepsilon_i \sim N(0,V_i)$ and in T is $\varepsilon_T \sim N(0,V_T)$, then the uncertainty ε_i' in P_i' is distributed $N(0,V_i')$ with

$$V_i' = V_i + V_T \tag{12}$$

That is, transforming the point to the current coordinate frame inflates its covariance by the amount of uncertainty in the transformation itself. In this we have overlooked some correlation induced by (11). Our initial assumption is that the errors in any landmark location are independent from all other landmarks. Equation (11), by applying the *same* uncertain transformation to all landmark locations, will cause the new coordinates P_i' to be correlated between landmarks [12]. Taking such correlations into account would increase the cost of the update quadratically for a small performance improvement, so we choose to ignore it.

Let the measurement of the landmark taken from the new robot location be Q_i with covariance V_{Qi}. We wish obtain an updated estimate of the landmark's coordinates by combining P_i' and Q_i. Treating these as two estimates of the mean and covariance of an unknown 3-D normal distribution and

Figure 5



applying standard linear statistical theory leads to the following updated estimates of the point location and its uncertainty:

$$V_{i+1} = (V_i'^{-1} + V_{Qi}^{-1})^{-1}$$

(13)

$$P_{i+1} = V_{i+1}(V_i'^{-1} P_i' + V_{Qi}^{-1} Q_i)$$

Recall that the V's are 3×3 covariance matrices. The intuition behind (13) is as follows. The elements of the covariance matrices V_i' and V_{Qi} will be large if the uncertainty of the corresponding estimates P_i' and Q_i is large. The larger the elements of a covariance matrix, the smaller (loosely speaking) will be the elements of its inverse. Hence, the more uncertain a measurement, the less weight it receives in estimating P_{i+1}. Laumond and Chatila [4] have described the analogous averaging scheme for scalar quantities.

Another way to formulate the point location update is to use Kalman filters. Taking Q_i as the new measurement and P_i as the state to be updated, we obtain [6]

$$V_{i+1} = (V_i'^{-1} + V_{Qi}^{-1})^{-1}$$

(14a)

$$P_{i+1} = P_i' + V_{i+1} V_{Qi}^{-1} (Q_i - P_i')$$

(14b)

V_{i+1} here is the same as in equation (6); furthermore, it can be shown that the estimates of P_{i+1} arrived at by (13) and (14) are identical. There is, however, a difference in the cost of the two formulations; using (13) requires three matrix-vector products and one vector-vector add, whereas (14) requires two matrix-vector products and two vector-vector adds. The latter is cheaper overall. The intuition behind (14b) is fairly simple. The second term takes the difference of the new measurement from the old state estimate $(Q_i - P_i')$, weights the difference by $(V_{i+1} V_{Qi}^{-1})$, and applies it as an update to the old state estimate P_i'. Matrix V_{Qi}^{-1} will be "larger" for more accurate new measurements, giving them more weight, and "smaller" for less accurate measurements, giving them less weight. Conversely, V_{i+1} will be "small" for an accurate old estimate, so that the new update is weighted less, and vice versa for an inaccurate old estimate. We have used the filter formulation of (14) in our implementation.

5. Updating the Global Robot Position

Previous sections have dealt with estimating each step of the robot's motion and updating the local world model. In this section we are concerned with estimating the robot's global position and positional uncertainty. This involves summing or integrating the step-wise motion estimates. Smith and Cheeseman [16] have recently shown how to do this for motion in the plane, involving two degrees of translation and one degree of rotation. They give the details of a Kalman filter formulation of the problem. Hallam [9] appears to have used a similar approach, although detailed equations are not shown. An extension to unconstrained, six degree-of-freedom motion has not yet appeared in the

computer vision and robotics literature. We will illustrate the approach for translational motion, summarize the Smith and Cheeseman treatment of planar motion, and discuss the difficulties with extending this to unconstrained motion.

Suppose that after i steps the robot's position is T_i with covariance V_i and that the next step is estimated to be T_s with covariance V_s. The new global position is

$$T_{i+1} = T_i + T_s \tag{15}$$

Since (15) is linear and involves gaussian variables, the error in T_{i+1} will be gaussian with covariance

$$V_{i+1} = V_i + V_s$$

The difficulty in extending this to motions involving rotation is that the update equation (15) is no longer linear, so the error propagation is no longer strictly gaussian. Smith and Cheeseman solve this for planar motion by linearizing. Each step-wise motion is represented by an uncertain translation (X,Y) in the floor plane and an uncertain rotation θ about the vertical axis. Given two such motions (X_1,Y_1,θ_1) and (X_2,Y_2,θ_2), they obtain closed form expressions for the variables X_3, Y_3, *and* θ_3 of the combined motion in terms of the variables $X_1, \ldots, \theta_2$. The equations are nonlinear and result in a non-gaussian distribution for the combined motion. They approximate this with a gaussian distribution obtained by linearizing. They also show how to use Kalman filter methods to incorporate motion estimates from several sensors into one overall position estimate.

When the motion involves all six degrees of freedom, the linearization approach is harder to apply because it is difficult to obtain closed form expressions for the combined motion in terms of the component motions. We speculate that expressing the rotation as a quaternion may lead to a manageable formulation. It seems likely that this problem has been addressed before in aerospace applications.

6. Simulation Results

A number of simulations were run to compare the performance of the 3-D normal error model to the performance of scalar weights. These experiments first examined the performance on a single step of robot motion, then the performance over several steps. The methodology attempted to mimick the configuration of cameras, objects, and motions used in our previous experiments with a real vehicle and real images [17]. The simulated cameras had a resolution of 512x512 pixels, a focal length of 12mm, and a field of view of 53 degrees. The baseline between cameras was 0.5 meters. The zscene consisted of random points uniformly distributed in a 3-D volume in front of the cameras. Typically this volume extended 5 meters to either side of the cameras, 5 meters above and below the cameras, and from 2 to 10 meters in front of the cameras. Image coordinates were obtained by projecting the points onto the images, adding gaussian noise to the floating point image coordinates, and rounding to the nearest pixel. These coordinates would be the input to the algorithms described above for triangulation, motion solving, and model update.

To obtain covariance matrices for point locations, image coordinates were assigned a distribution with standard deviations of one pixel for each of x_l, y_l, x_r, y_r and no correlation between any two coordinates. These were propagated through the triangulation as described in section 2. Scalar weights were derived by taking the Z variance from the covariance matrix. Scalars obtained by several other methods were tried and found to give very similar results. These include the volume and length of the major axis of the standard error ellipsoid and Moravec's half-pixel shift rule [13].

6.1. Single step motion

Planar motion estimation was tested first. After a step of one meter directly forward, the robot estimated its lateral translation (X axis), forward translation (Z axis), and rotation about the vertical (Y axis). Experiments were done varying the number of points tracked and the distribution of the points in space. For any one experiment, averages and standard deviations were calculated for the results of 5000 trials. In this set of simulations no noise was added to the image coordinates, so that quantization of the image was the only source of error.

When all points were 2 to 10 meters away, which corresponds to disparities of 13 to 64 pixels (roughly 3% to 11% of the image width), the mean estimate of the forward motion was within 0.1% of correct for both scalar and matrix weights and for anywhere from 6 to 50 points tracked. Since the true motion was 1 meter, this implies average estimates of about 0.9995 meters. The error that did occur showed a slight bias to underestimate the true motion.

Standard deviations of the motion estimates as a function of the number of points tracked are plotted in figures 6 and 7. Figure 6 shows the results for rotation. Estimates based on scalar weights have about 10 times the spread of estimates based on matrix weights. With 20 points tracked, the standard deviation with matrix weights is about 0.03 degrees. Figure 7 shows the results for X and Z translations. There is a factor of 10 difference in spread between the scalar and matrix cases for X, but only a factor of 5 for Z. This is explained by the fact that lateral translations and vertical rotations have a coupled effect on errors of fit, so that small lateral translations strongly resemble small rotations about the vertical axis. It is significant that the coupling is reduced by using matrix weights. With matrix weights, tracking 20 points yields standard deviations in X and Z of approximately 0.004 meters over a 1 meter motion, or 0.4% of the distance travelled. This compares to 3.5% and 1.9% for X and Z, respectively, with the scalar algorithm.

The results for motion solving in space are comparable to the results for motion in the plane. The previous experiment was re-run with the motion solver estimating all six parameters of motion. The average solution for Z translation still underestimated the true motion by about 0.1%. Figures 8 and 9 show the standard deviations of the rotations and the translations, respectively. The pattern is very similar to the three degree of freedom case. The deviations are roughly the same size and the ratios between scalar-based and matrix-based motion solving are the same. The scalar-based algorithm shows a coupling between lateral translation and panning rotation, vertical translation and tilting rotation, but not between forward translation and rolling rotation. Using full covariance matrices moderated this effect.

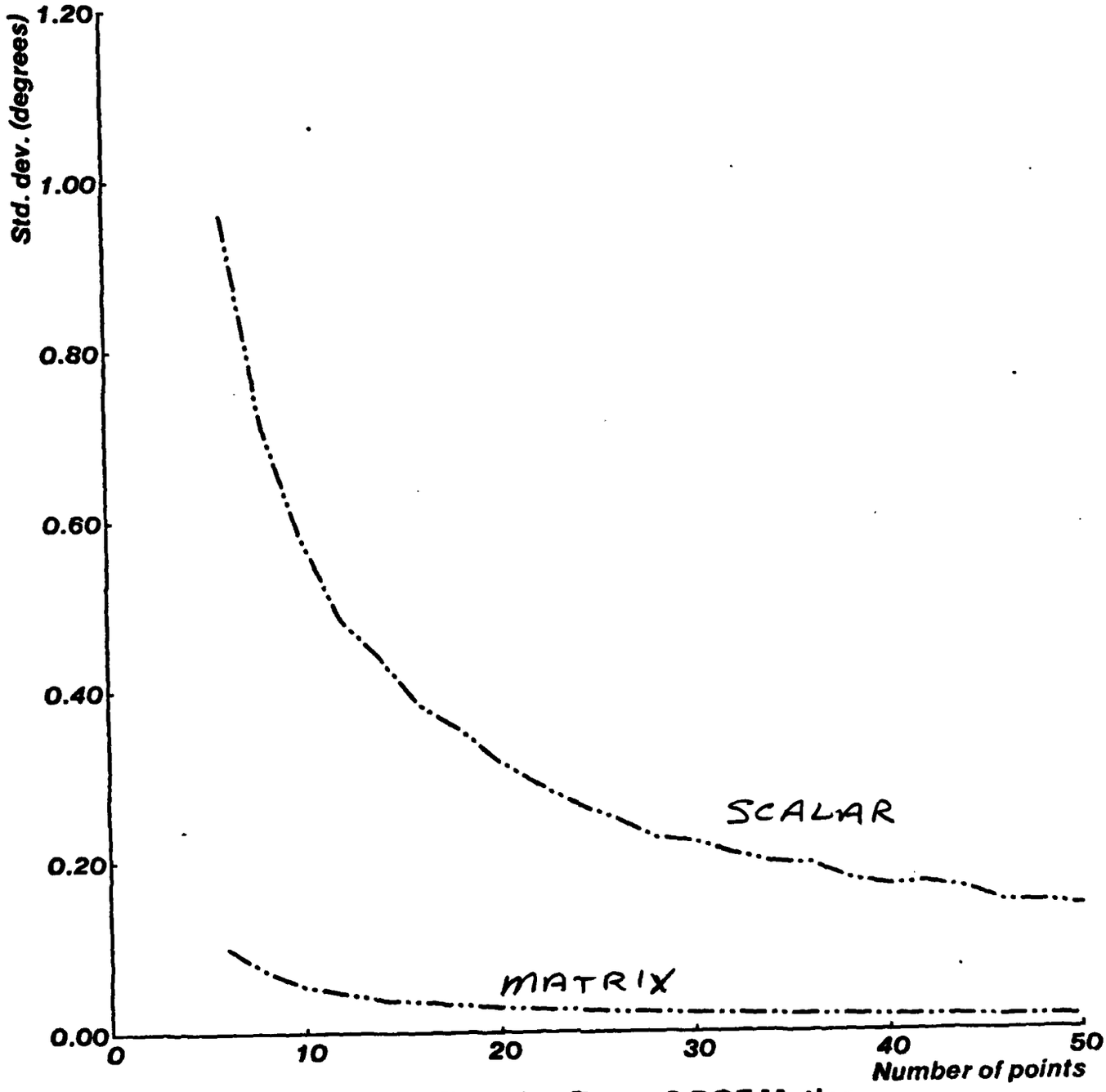


Figure 6: Y Rotation Error, 3 DOF Motion

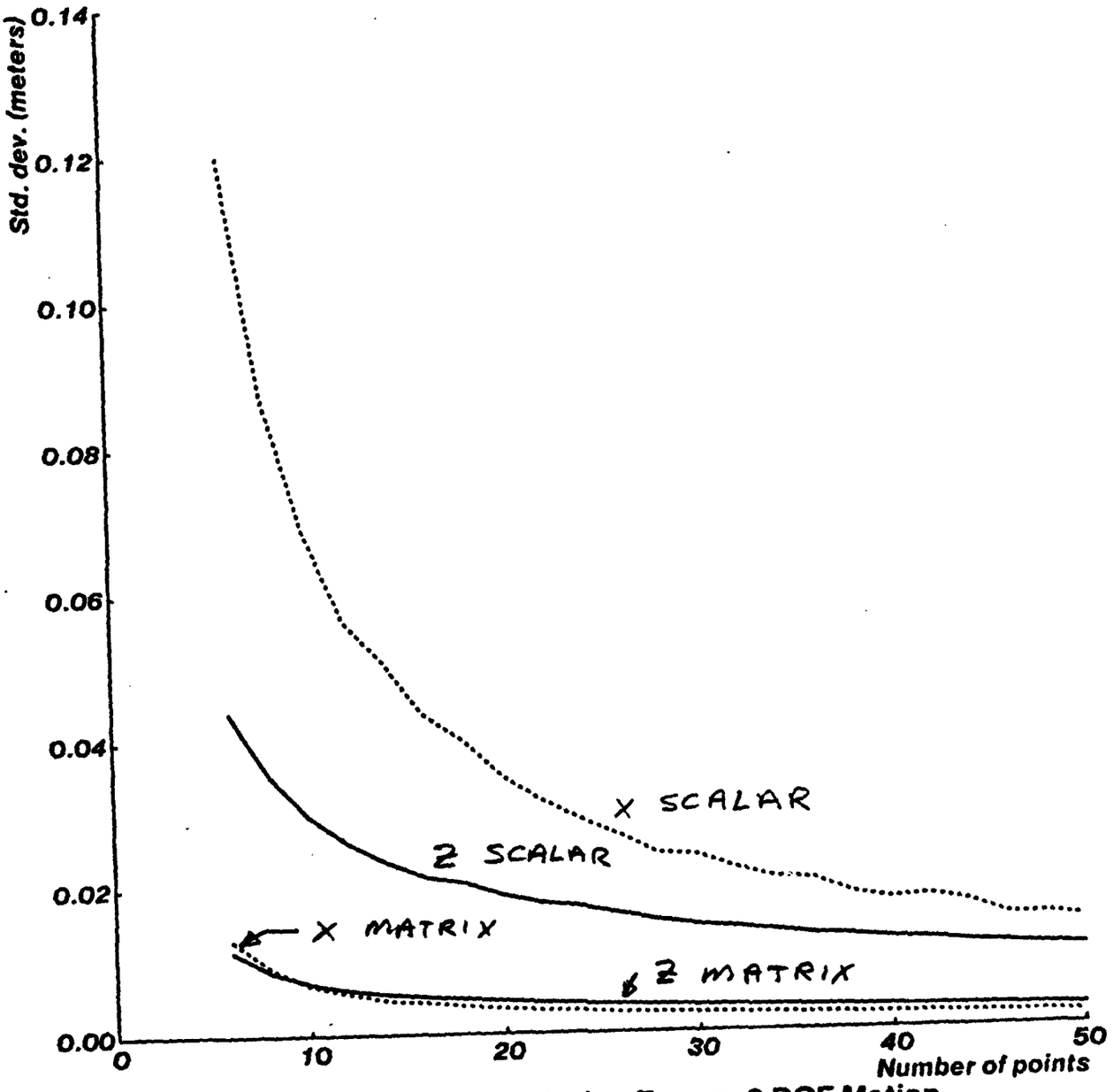


Figure 7: X and Z Translation Errors, 3 DOF Motion

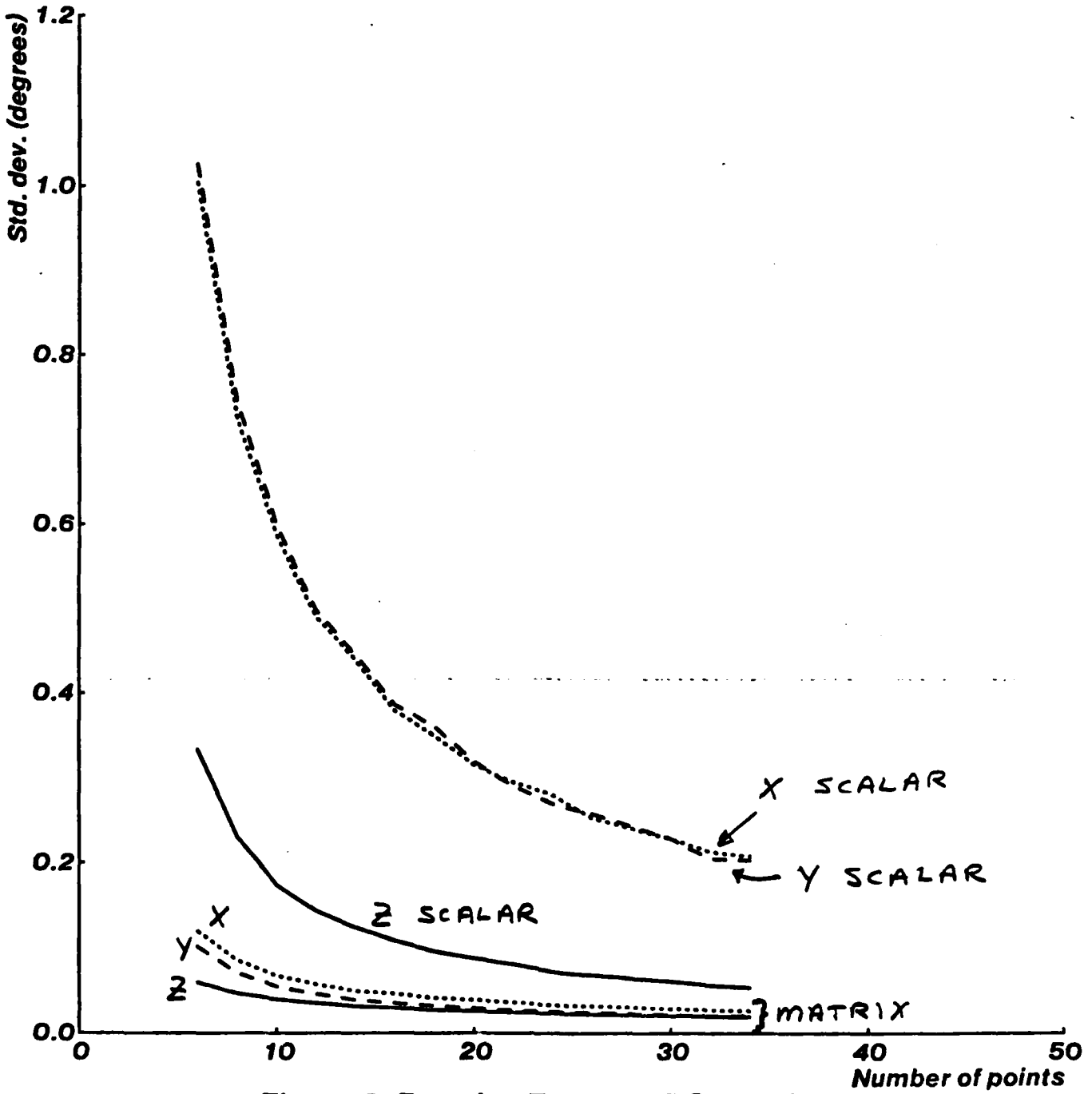


Figure 8: Rotation Errors, 6 DOF Motion

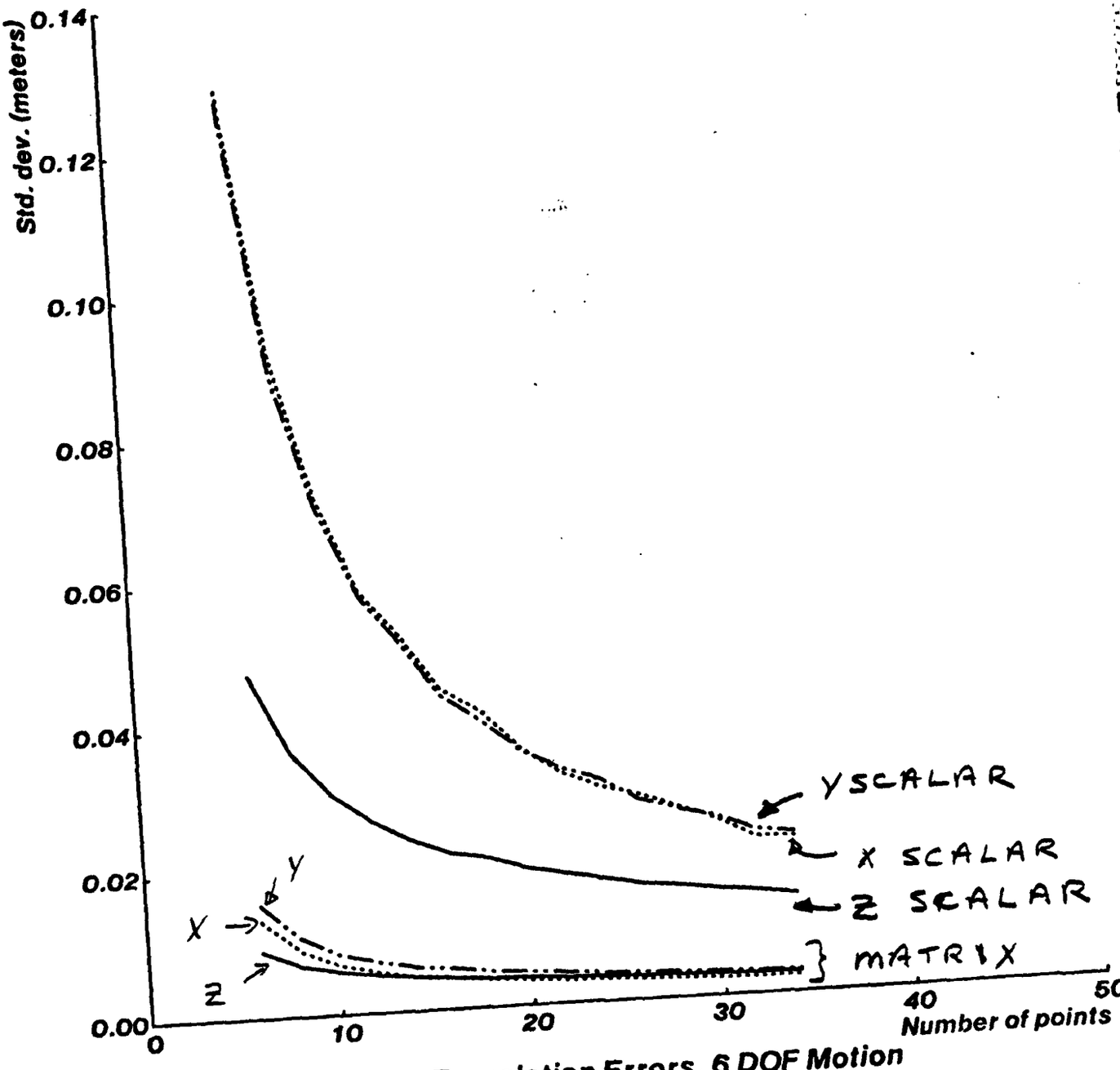


Figure 9: Translation Errors, 6 DOF Motion

Adding gaussian noise to the image coordinates in both of these experiments increased the standard deviations for all curves, but did not effect the ratios between scalar and matrix results.

The next experiment tested the effect of increasing the distance to points in the scene, or equivalently reducing the maximum disparity in the image. Twenty points were generated in a volume spanning 4 to 50 meters in front of the cameras (see inset of figure 10), giving disparities ranging from 2 to 32 pixels or 0.5% to 6% of image width. The volume was gradually shrunk by moving the near limit from 4 meters back until all points were 50 meters away, so that all disparities were on the order of 2 to 3 pixels. The results are shown in figures 10 and 11. The horizontal axes in both figures show the distance to the near limit and the equivalent pixel disparity in the 512x512 images. Figure 10 shows the average, over 5000 trials, of the estimated forward motion as a function of maximum disparity. The matrix-based method underestimates the motion by 1% to 2% for all disparity ranges. The scalar-based method, on the other hand, underestimates by about 1% when large disparities are available, but overestimates by close to 30% when all disparities are near 3 pixels. The standard deviations of these estimates are shown in figure 11. The spread is tight for all disparity ranges with the matrix-based algorithm, but grows rapidly with shrinking disparity with the scalar-based algorithm. The breakdown with distance shown by the scalar algorithm is well-known in computer vision; this makes the stability of the matrix algorithm come as quite a surprise.

6.2. Multi-step motion

Motion over several steps was simulated to examine the behavior of the point location update and the effect it had on the accuracy of successive motion estimates. The simulated robot moved one meter straight ahead between each stereo pair. Twenty points were used to solve for motion each time. These were initially generated in the volume 2 to 10 meters in front of the robot. As points fell out of the field of view, new points were generated in the volume 2 to 10 meters in front of the current robot position. The unknowns in the motion solver were just the three components of translation. In this experiment gaussian noise with a standard deviation of half a pixel was added to the image coordinates before rounding to the nearest pixel.

To test the effect of the point update, a point initially 15 meters in front of the robot, 4 meters to the right, and at eye level was tracked until it passed out of view. The mean location for both scalar and matrix methods was within 5 centimeters or 3% at all times, with the error being an underestimate of the true distance. The standard deviations of the robot-centered coordinates of the point after each step are shown in figure 12. As expected, averaging more measurements causes the standard deviations to shrink. Note that after six steps (seven sightings), spread of the matrix-based update is half that of the scalar-based update.

Successive estimates of the robot motion showed a steady bias of about 0.5% on the short side for both algorithms. That is, the one meter motions were always estimated to be about 99.5 centimeters. Figure 13 shows the standard deviations of these estimates over time. The spread of the estimates shrinks until it reaches a plateau. The matrix-based estimates have one third to one quarter the variability of the scalar-based estimates. This showed up in the accumulated global position

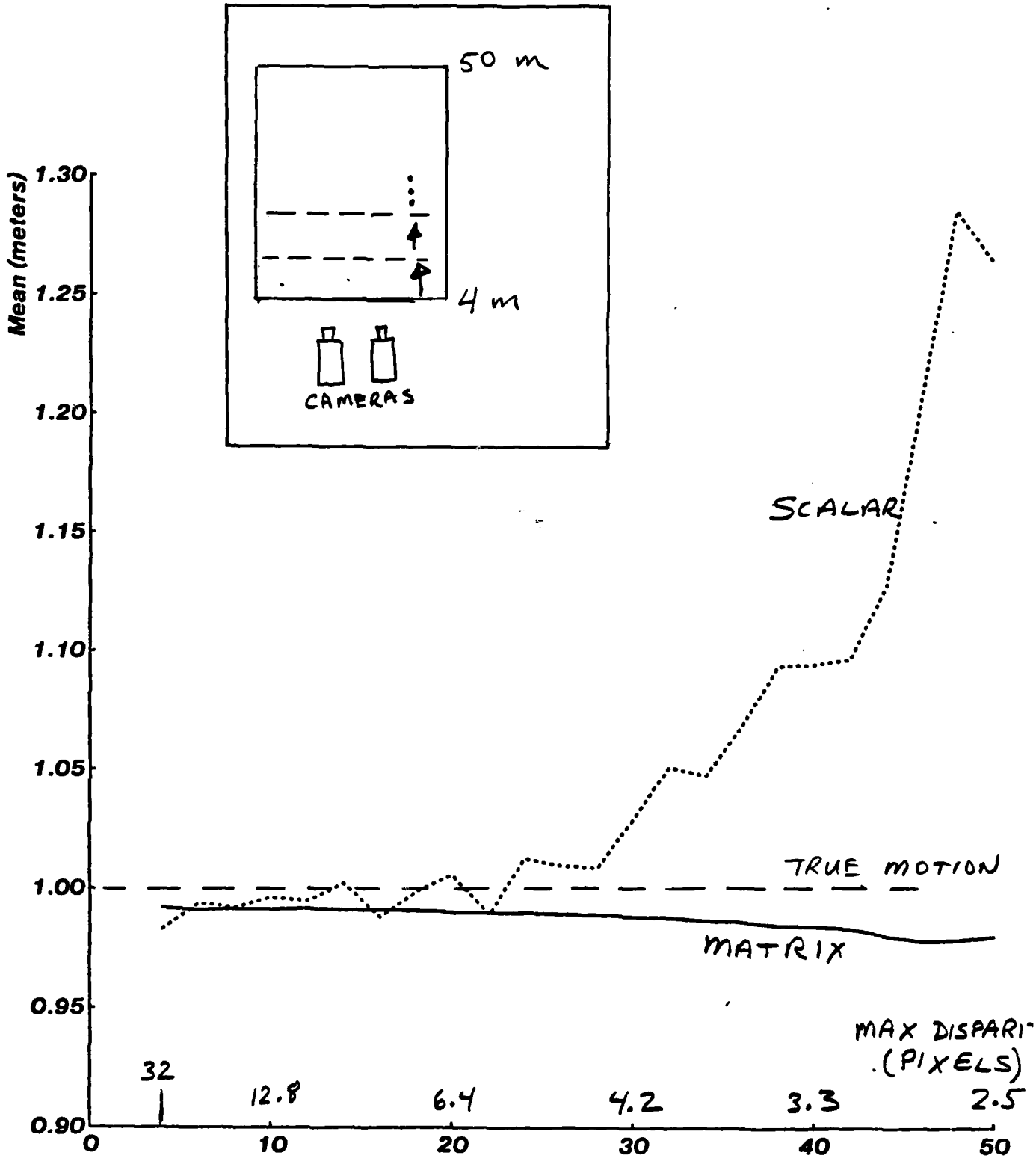


Figure 10: Motion Bias (true = 1.0) vs. Point Distance

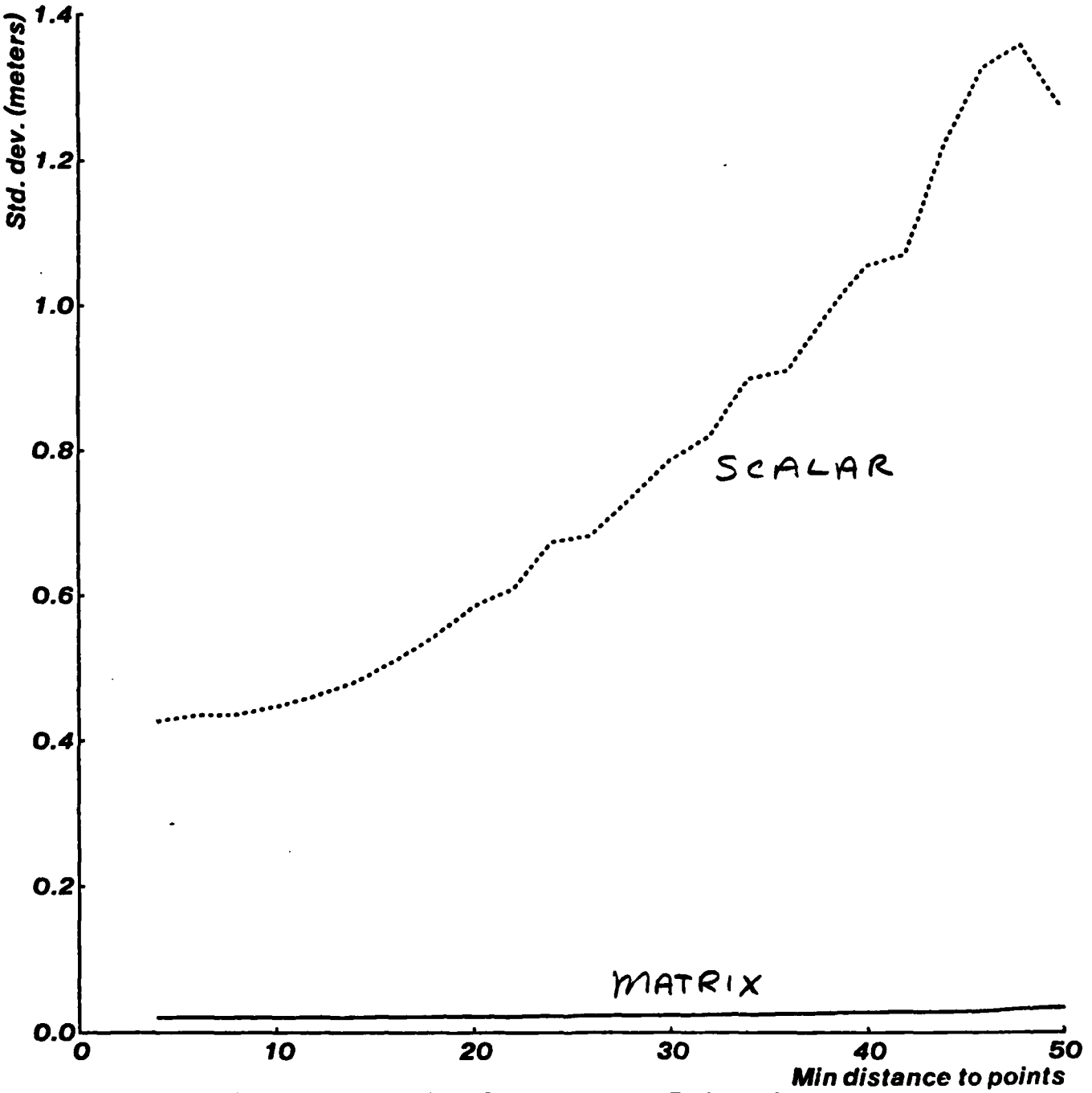


Figure 11: Motion Std. Dev. vs. Point Distance

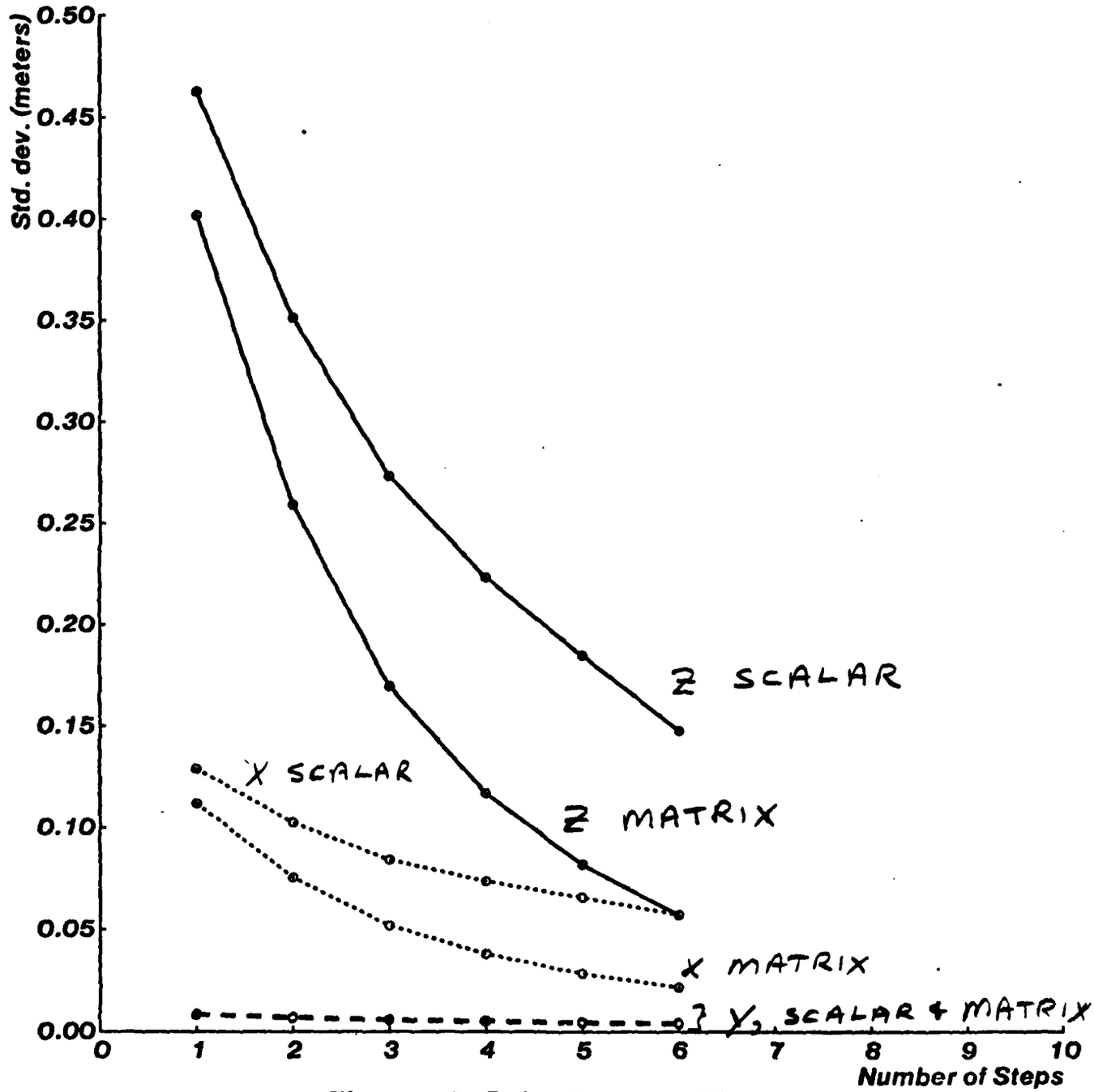


Figure 12: Point Error vs. Time

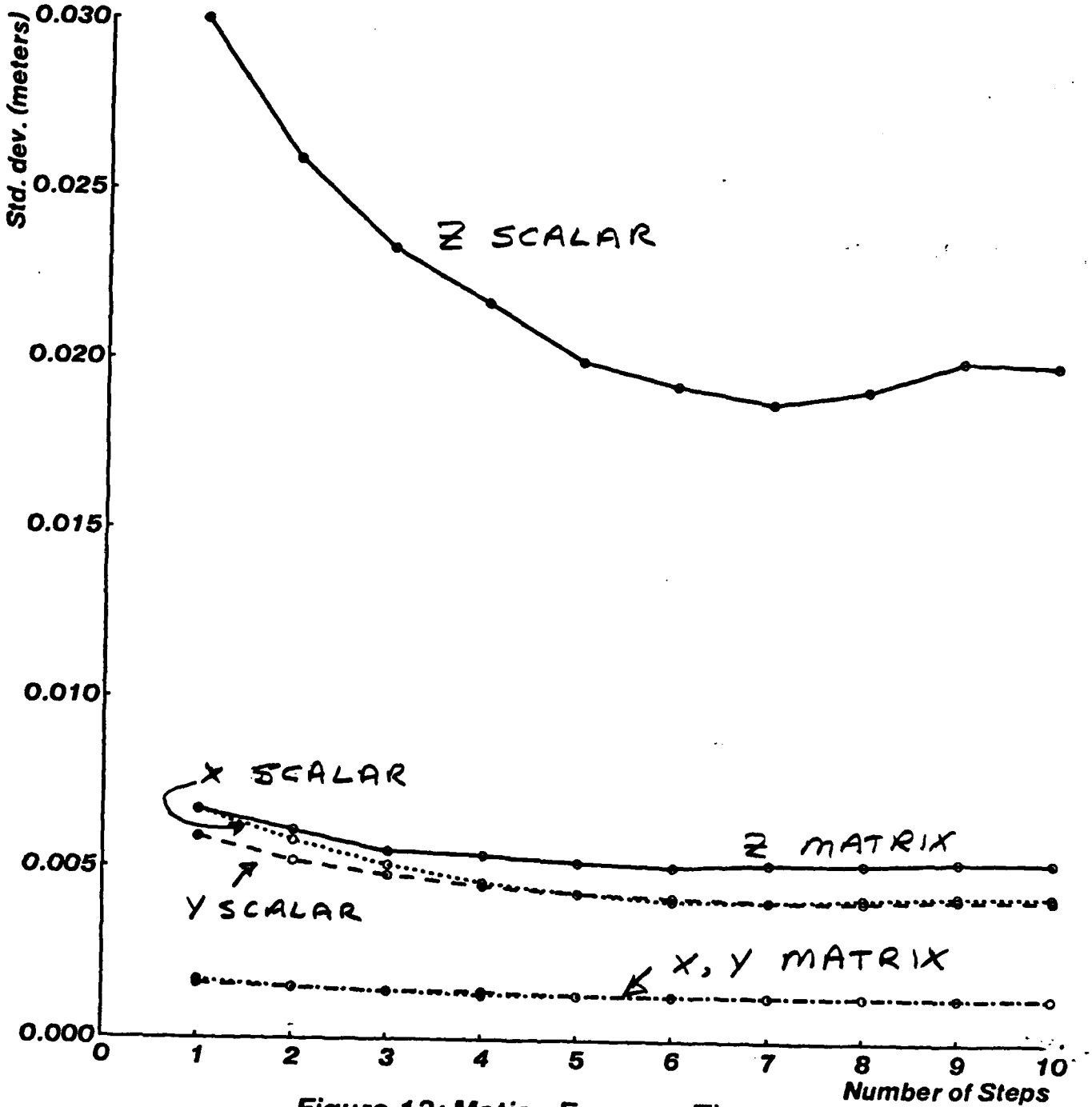


Figure 13: Motion Error vs. Time

estimates. After 10 meters of simulated motion, the standard deviation of the matrix-based estimates of total distance travelled was 1.6 cm, versus 4.4 for the scalar method.

7. Discussion

The goal of this paper was to show that using a model of stereo triangulation error based on 3-D normal distributions would lead to more accurate motion estimation than scalar error models. The simulations have verified this claim. Step-wise motion estimates, global position estimates, and landmark location estimates are better with the new method than the scalar method. Other motion solving algorithms from the literature [2], not based on probabilistic error models, had performance to our scalar-weighted algorithm and poorer than the matrix-weighted version.

Three dimensional normal distributions model triangulation error better than do scalars, but they are not entirely faithful to reality either. This shows up in the biased estimates obtained in the simulations. However, these biases are small enough that it may be acceptable to ignore them.

One of the most striking aspects of the new model is the improved performance is gives with distant points. This implies that the new method permits shorter stereo baselines to be used without sacrificing accuracy of the motion estimate. Since the length of the baseline directly affects the difficulty of stereo matching, this may offer a way to alleviate the correspondence problem.

Our first priority for future work is to verify the simulation results with tests on real images. Should the results hold up on data free of correspondence errors, the next step will be to pursue the idea of shortening the baseline to reduce the likelihood of mismatches. This will be augmented with statistical tests to filter any remaining mismatches. Further extensions include coping with general rotation in the global position update, tracking lines as well as points, and estimating velocity as well as position.

8. Acknowledgements

This work has been supported at the Carnegie-Mellon University Robotics Institute since 1981 by the Office of Naval Research under contract number N00014-81-K-0503.

Hans Moravec, Steve Shafer, and Takeo Kanade have provided valuable comments and guidance during the course of this work. Takeo observed that the results might enable the use of shorter stereo baselines. Peter Highnam brought the work of Schonemann to my attention.

References

[1] H.S. Baird.
Model-based image matching using location.
MIT Press, Cambridge, Mass., 1985.

[2] S.D. Blostein and T.S. Huang.
Robust algorithms for motion estimation based on two sequential stereo image pairs.
In *Conf. on Computer Vision and Pattern Recognition*. 1985.

[3] T.J. Broida and R. Chellappa.
Estimation of motion parameters from noisy images.
IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-6(1):90-99, January, 1986.

[4] R. Chatila and J.-P. Laumond.
Position referencing and consistent world modelling for mobile robots.
In *IEEE Int'l Conf. on Robotics and Automation*, pages 138-145. IEEE, March, 1985.

[5] T.F. Elbert.
Estimation and control of systems.
Van Nostrand Reinhold Co., New York, NY, 1984.

[6] A. Gelb (editor).
Applied Optimal Estimation.
MIT Press, Cambridge, MA, 1974.

[7] D.B. Gennery.
Modelling the environment of an exploring vehicle by means of stereo vision.
PhD thesis, Stanford University, June, 1980.

[8] A.W. Gruen.
Algorithmic aspects in on-line triangulation.
In *XVth Int'l Congress of Photogrammetry and Remote Sensing, Commission III, Part 3a*, pages 342-362. Int'l Society for Photogrammetry and Remote Sensing, 1984.

[9] J. Hallam.
Resolving observer motion by object tracking.
In *Int'l Joint Conf. on Artificial Intelligence*. 1983.

[10] M. Hebert.
Reconnaissance de formes tridimensionelles.
PhD thesis, L'Universite de Paris-Sud, Centre d'Orsay, September, 1983.

[11] L.H. Matthies and C.E. Thorpe.
Experience with visual robot navigation.
In *IEEE Oceans'84*. IEEE, Washington, D.C., August, 1984.

[12] E.M. Mikhail.
Observations and Least Squares.
University Press of America, Lanham, MD, 1976.

[13] H.P. Moravec.
Obstacle avoidance and navigation in the real world by a seeing robot rover.
PhD thesis, Stanford University, September, 1980.

[14] P.H. Schonemann and R.M. Carroll.
Fitting one matrix to another under choice of a central dilation and a rigid motion.
Psychometrika 35(2):245-255, June, 1970.

[15] C.C. Slama (editor-in-chief).
Manual of photogrammetry.
American Society of Photogrammetry, Falls Church, Va., 1980.

[16] R.C. Smith and P. Cheeseman.
On the representation and estimation of spatial uncertainty.
Technical Report, SRI International, 1985.

[17] C.E. Thorpe.
Vision and navigation for a robot rover.
PhD thesis, Carnegie-Mellon University, December, 1984.

Road Following

First Results in Robot Road-Following

Richard Wallace, Anthony Stentz

Charles Thorpe, Hans Moravec

William Whittaker, Takeo Kanade

Robotics Institute, Carnegie-Mellon University

Abstract

The new Carnegie-Mellon Autonomous Land Vehicle group has produced the first demonstrations of road-following robots. In this paper we first describe the robots that are part of the CMU Autonomous Land Vehicle project. We next describe the vision system of the CMU ALV. We then present the control algorithms, including a simple and stable control scheme for visual servoing. Finally, we discuss our plans for the future.

Introduction

CMU has formed the Autonomous Land Vehicle (ALV) group to develop a perceptive outdoor robot. We have produced the first demonstrations of an autonomous vehicle able to follow a road using a single on board black and white television camera as its only sensor. Our robot has made several successful runs over a curving 20 meter path, and 10 meter segments of straight sidewalk, moving continuously at slow speeds, by tracking the edges of the road.

The research described in this paper is a first complete system, covering everything from low-level motor drivers to the top-level control loop and user interface. We took a "depth-first" approach to building our testbed: we picked one rough design and built all the pieces of a functioning system, rather than spending a lot of time at the beginning exploring design alternatives.

Related research at the University of Maryland [6] has focused on the problem of visually finding and tracking roadways. The "bootstrapping" phase of the Maryland road finding program, in which the robot detects a road on start-up with no a priori position information, currently has no counterpart in our system. Our vehicle is always started with an orientation more or less aligned with the direction of the road and with knowledge of an initial road model. The Maryland road finding module is expected to be tested soon on an ALV built at Martin Marietta Denver Aerospace.

In this paper we first describe the robots that are part of the CMU Autonomous Land Vehicle project. We next describe the vision system of the CMU ALV. We then present the control algorithms, including a simple and stable control scheme for visual servoing. Finally, we discuss our plans for the future.

Terregator and Neptune

No mobile robot system is complete without a mobile robot. The primary vehicle of the CMU ALV project is the Terregator, built in the Civil Engineering Department. The design and construction of the Terregator (for *terre*strial navi*gator*) is documented in [7]. It is a 6-wheeled vehicle, 64 inches long by 39" wide by 37" tall. All wheels are driven, with one motor for the 3 left wheels and one for the 3 right wheels. Shaft encoders count wheel turns, but the vehicle skid-steering introduces some indeterminacy.

The Terregator is untethered. Power is provided by an on-board generator. Communications with a host computer are via a bi-directional 1200 baud radio link for vehicle status and commands, and a 10 megahertz microwave link for television signal from the vehicle to a digitizer. A remote VAX 11/780 runs programs for symbolic processing of visual data and navigation. A Grinnell GMR 270 attached to the Vax computes low- level visual operations such as edge detection. A Motorola 68000 on the Terregator translates steering commands from the VAX into wheel velocities for the left and right wheels.

Earlier work also used the tethered robot Neptune, built by the Mobile Robot Lab. Neptune is a simple tricycle, with a powered and steered front wheel and two passive wheels in the rear. Its sensors consist of two cameras (for stereo vision work), plus a ring of 24 sonars. While it was intended primarily for indoor work, it has large enough wheels to run outside on gentle terrain. With suitable modifications (an umbrella taped to the camera mast), it even has limited all-weather capability.

Our first successful continuous motion road following was achieved with Neptune running in our lab on a road marked with black electrical tape on the floor. This 5 meter road had one left turn and one right turn, which Neptune navigated successfully. At the end of the road, Neptune made a sharp right turn and drove around in circles.

Currently, this project is funded in part by Carnegie-Mellon University, by the Office of Naval Research under contract number N00014-81-K-0503, by the Western Pennsylvania Advanced Technology Center, by Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539, and by Denning Mobile Robotics, Inc. Richard Wallace thanks NASA for supporting him with a NASA Graduate Student Researchers Program Fellowship Grant.

IJCAI-85

The Vision and Navigation Program

The primary task of our vision and navigation program is to keep the vehicle centered on the road as it rolls along at a constant speed. The program accomplishes this task by repeatedly digitizing road images, locating the road edges in the image, calculating the deviation from the center line, and steering to realign the vehicle.

The program was designed to be fast yet reliable. While the vehicle is moving along a planned path, an image is digitized.

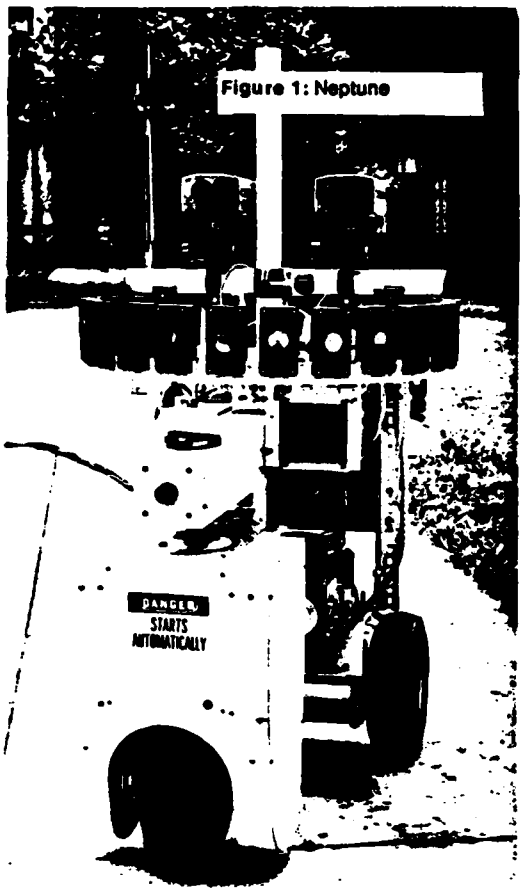


Figure 1: Neptune

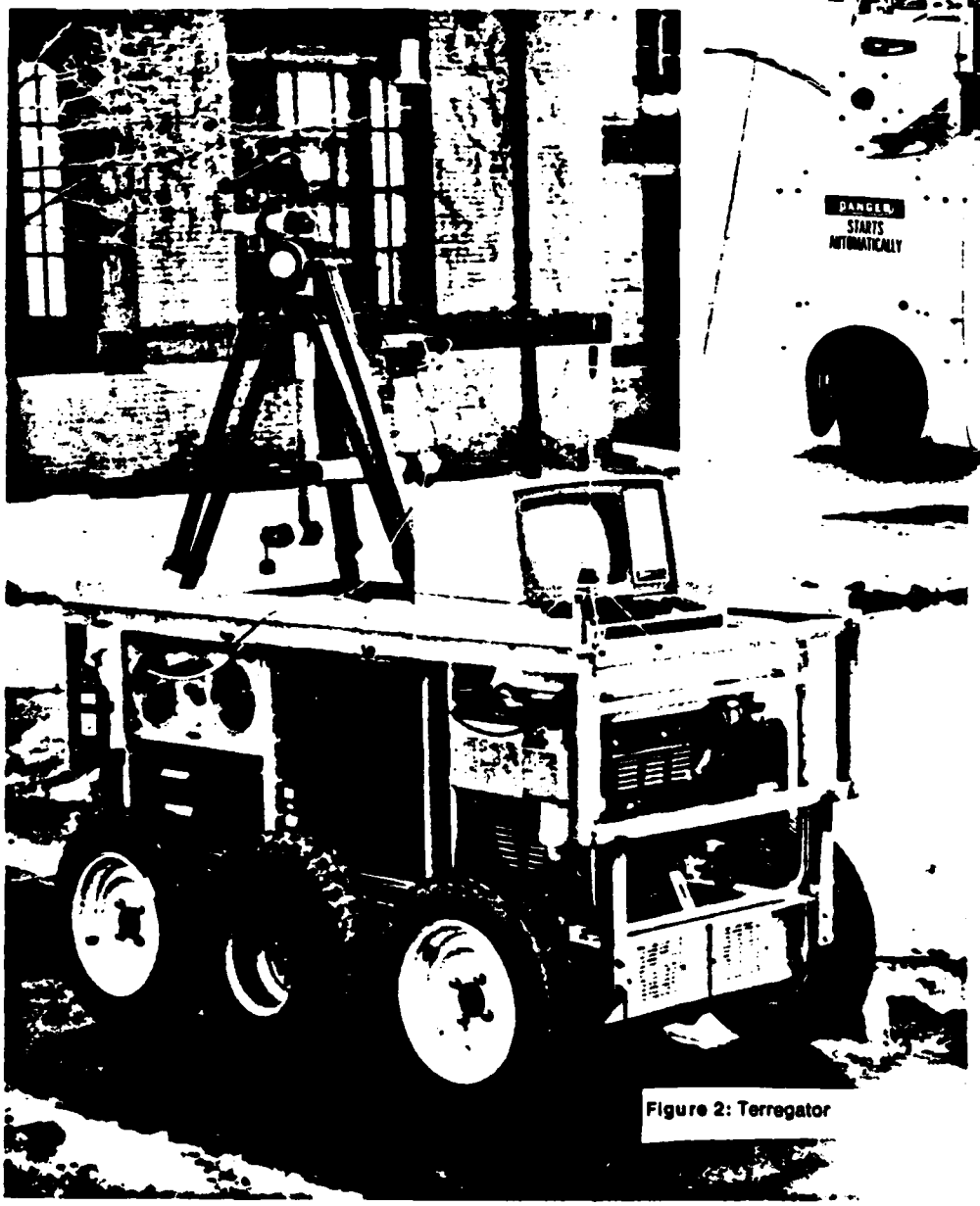


Figure 2: Terregator

Since images are digitized frequently, the appearance of the road edges does not change appreciably across successive images; consequently, searching the entire image is unnecessary. In order to constrain the search, the program maintains a model of the road The model contains the position and orientation of the left and right road edges seen in a recent image. The program uses these model edges to generate two small subimage rectangles in which to search for the left and right road edges. Since the approximate direction of each road edge is known a priori, the program uses directed curve tracing to reduce processing time and to preclude spurious edges. Generally the program finds more than one edge in each subimage rectangle. The model is used to select the pair of extracted edges most likely to be road edges. This new pair replaces the old pair in the model. From the model pair, the program computes a center line, the vehicle's drift from the center line, and a steering command to bring the vehicle closer to the center line. As the vehicle executes a steering command another image is digitized and the cycle repeats. Figure 3 depicts the program control flow. In the remainder of the paper we explain each component of the program in greater detail.

Constraining the Search

Each time the program digitizes an image it chooses two subimage rectangles to constrain the search for left and right edges. The representation of the rectangle is two horizontal and two vertical bounding line segments. The vehicle always "looks" a fixed distance ahead; therefore, the placement in the image of the horizontal bounding segments is predetermined and remains fixed across successive images. The placement of the segments is partly determined by two parameters selected manually: the height of the rectangle (typically 50 to 100 pixels) and rectangle overlap, that is, the percentage of the road in a rectangle seen in the preceding image (typically 50%). These two parameters present important trade offs: If a large height is chosen, the extracted road edges will be longer, thus providing more accurate information about the road; however, the processing time will be increased, and the road will be scrutinized less often. If a large overlap is chosen, more information is available from the previous image and spurious edges are less likely to deceive the algorithm; however, the vehicle's speed must be slowed to enable such overlap. The two parameters, coupled with the vehicle's speed, the image processing time, and the camera's tilt determine the placement of the horizontal bounding segments in the image.

The vertical bounding segments change from image to image. The program selects bounding segments so that the road edges, based on predictions from the model and a preset error tolerance, will appear within the rectangle. This error tolerance arises from two sources: First, the program obtains its estimates of the vehicle's motion by dead reckoning, which is somewhat inaccurate. Second, the program assumes the road is straight, that is, predictions are made by linearly extending the road edges. Road curvature introduces a discrepancy between these predictions and the actual road; consequently, the rectangle must be wide enough to see the road edge within a preset tolerance.

Selecting the Best Edges

The line finding routine generally returns more than one line from each rectangle. The program passes these lines through a number of filters to determine which, if any, are road edges. The new road edges are used to plan a path for the vehicle and to update the model. The 16 best left and right edge candidates (based on weights supplied by the line finding routine) are retained, and the rest are discarded. The program assumes that the camera's calibration, position, and orientation with respect to the road are known, that the ground is locally level and that all candidate edges arise from ground features. These assumptions allow the program to project each candidate edge into a unique line in the ground plane. We establish a righthanded coordinate system with the vehicle at the origin and the xy-plane on the ground, with the positive x-axis directed to the right of the vehicle and the positive y-axis directed forward. For each transformed edge, the program calculates the following parameters: the perpendicular distance r measured from the origin to the edge and the angle θ measured from the positive x-axis. The differences in r and θ between each transformed candidate edge and the corresponding model edge are calculated (call these values dr and $d\theta$ respectively). The quantity dr is the difference in displacements of the vehicle from the model edge and from the candidate edge. The quantity $d\theta$ is the angle between the model edge and the candidate edge. Test runs have shown that the vehicle tends to remain aligned with the center line; most of the error is in the form of lateral drift from this line. Hence, dr provides the most information for evaluating candidate edges. The quantity $d\theta$ tends to be small (less than 10 degrees); consequently, an early filter uses it to eliminate spurious edges. After this round of edge elimination, one of three cases remains:

1. All edge candidates have been eliminated
2. All edge candidates have been eliminated for a particular road edge (either left or right)
3. At least one edge candidate remains for both the left and right road edge

In the first case, the program obtains no new information and the vehicle continues to execute the path planned from the previous image. In the second case, only one road edge is visible. The other road edge is occluded, shadowed, or poorly defined. Suppose for example the program found a set of candidate road edges on the right side but none on the left. From the candidate edges on the right side the program selects the one with the minimum dr value. It inserts this new edge into the model, retains the old model edge for the left side, and generates a new steering command. In the third case, both road edges are visible. The program selects one edge from each list of road edges (left and right) by comparing each left edge to each right edge candidate and choosing the pair that minimizes the difference in their dr values, that is, it selects the two edge candidates that differ from their corresponding model edge in the same way. Figure 3 illustrates road edge selection in this case. This decision is based on the observation that vehicle motion error and road curvature shift the location of each edge in the image in the same way. The program inserts the two new road edges into the model and plans a new path.

Line and edge extraction

At the lowest levels of the vision system for our vehicle, the edge and line extraction modules, we found that for detecting road edges we could rely on the principle "almost anything works in the simple cases." That is, any of a number of simple edge and line finding techniques could be used to extract road edges in various situations. Our approach then was to try everything. We tested various edge and line finding programs on static road images and on images acquired by the vehicle in actual runs. Simple techniques proved adequate in many situations we encountered.

The basic approach of all the vision modules we tried was to find the left and right boundaries of the road and represent them as lines. Therefore, the task of the low level vision modules is to find line segments which are plausible candidate road edges. We sought to make only the most general assumptions about what might constitute a road in an image. The technique used to extract road edges and represent them as lines depends on

whether we think of a road as an intensity change from background, a texture change, a color change or a combination. We experimented with 7 methods for extracting road edges from images and three methods for fitting lines to the edges. The seven techniques we used to find edges in road images were:

1. **Correlation.** Assuming that a road edge is a more or less vertical feature in a subimage it can be followed by selecting a small sample patch of the edge and correlating this on a row-by-row basis with the subimage. Where the correlation is strongest in each row a road edge element is assumed. The result is a list of points where the road edge appears in each row. A line can be fit to these directly. The correlation approach worked very well when the sample road edge patch was hand selected.

2. **DOG operator.** A Difference of Gaussian edge operator was tried at a wide range of spatial resolutions on road images. Road edges tend to be low spatial frequency signals so large DOGs were required to find them directly. Two-dimensional DOG filters tended to break up the road edges even at low frequencies. One dimensional DOG operators applied horizontally in the image produced more connected road edge pieces, since the road boundaries were almost vertical features in the image. High spatial frequency DOG operators can be used as the basis of a texture-based segmentation of road images, however.

3. **Temporal Edge Detector.** Subtracting two successive image frames is an inexpensive method for detecting image features that change from one moment to the next. If a vehicle is traveling down an ideal road (where the intensity of the road is uniform, the intensity of the surrounding region is uniform and the road edges are straight and parallel) then the difference of two successive road images is zero. When the vehicle begins to turn left or right off the road, however, simple image differencing finds the road edges. This strategy was used in one experiment to servo Neptune visually down a hallway. Here the road edges were particularly distinct so the idealness assumption was more or less satisfied.

4. **Roberts Operator.** A 2x2 Roberts edge operator was sufficient to find road edges where they were relatively well-defined intensity step functions, such as when the vehicle traveled down a hallway or when we artificially marked the road edges with tape.

5. **Intensity Segmentation.** A simple binary intensity segmentation of the road image works in many cases where the road is a set of pixels most of whose intensities are grouped together in the image histogram. We used a simple segmentation technique based on classifying all the pixels in the bottom 50% of the histogram as one region and those in the upper 50% as another. Standard procedures for expanding and shrinking the resulting segments to join closely spaced segments and eliminate small ones are applied. Road edges are assumed to lie along the boundaries of the resulting regions.

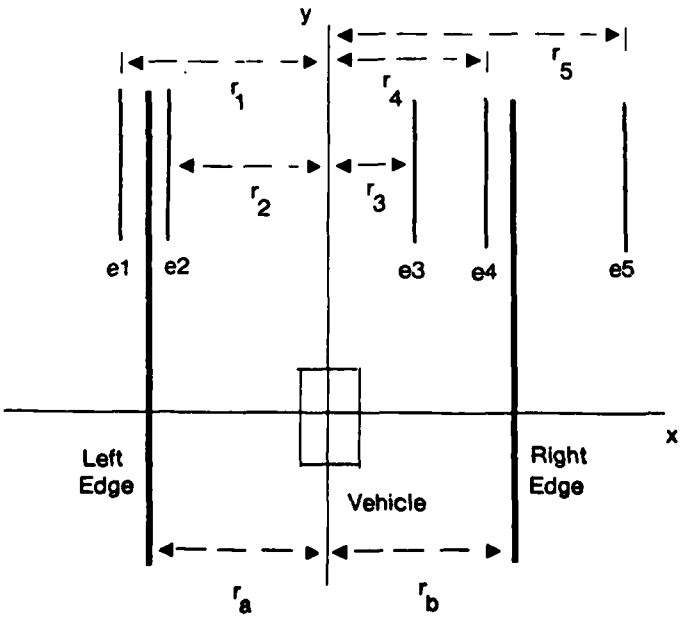


Figure 3: Edge selection using the perpendicular distances. Only edge candidates with $\theta = 0$ were included for simplicity. Candidates e1 and e4, with $r_{left} = r_1$ and $r_{right} = r_4$, minimize the error $|(r_{left}-r_a)\ (r_{right}-r_b)|$ and are selected as the new model road edges.

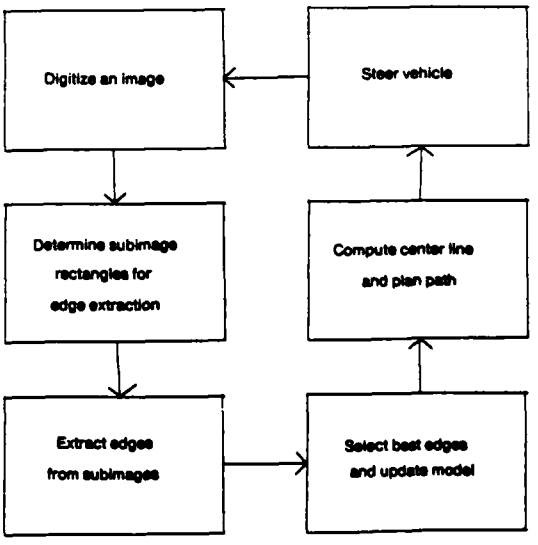


Figure 4: System Block Diagram

6. **Texture Segmentation.** Texture based segmentation often proves better than intensity based segmentation for road edges where the road is relatively smooth and the surrounding region is not, such as when the road is asphalt against a grass background. A simple texture operator which we have found useful in detecting road edges is one which counts the number of edges per unit area and classifies all those areas where the edge count is high as a single region.

7. **Row Integration.** Summing the intensities column-by-column in a set of scanlines in the image results in a single-scanline intensity image where the road is roughly a one dimensional box function, given that the road is a more or less vertical feature and the road and surrounding area each have fairly uniform but different intensities. Finding the boundaries of the box amounts to finding the average position of the left and right road edges over the scanlines summed. Repeating the procedure for another set of rows in the image locates another pair of road edge points which can be joined with the first to approximate the road boundaries as line segments.

The three line-extraction techniques we used were:

1. **Least Sqaures Line Fitting.** When we had only one possible line in an edge image, such as the result of running a correlation operator over the rows or collecting a number of road edge points by row integration, a line could be fit to the points by least squares.

2. **Muff Transform.** A modified Hough (Muff) transform was used to fit lines to edge data where the edge extractor returned points that could plausibly be parts of several lines. The Hough transform has been used to detect road edges in other road finding programs [6] [1]. The Muff transform uses a new parameterization for lines in images. The Muff transform has several implementational advantages over the conventional ρ-θ parameterization. The details and implementation of the Muff transform are presented elsewhere [5].

3. **Line Tracing.** Most of the subimages we processed to find lines were bands about 50 pixels tall and 250 pixels wide. A simple raster tracking algorithm found in [3] proved sufficient to trace the road edges. Basically, if an edge point P above some high threshhold d is found while scanning the subimage, then we search on scan lines below for connected edge points above some lower threshhold t. The last such point found in the subimage is called Q and we assume PQ is a line segment. The line tracing procedure is much like the inverse of a Bresenham algorithm for drawing lines, with the similar limitation that we can find lines that are only with 45 degrees of vertical. We find lines more than 45 degrees from perpendicular and lines with gaps by searching in a neighborhood below an edge point for the next adjacent edge point. Strictly speaking, our tracing program returns the endpoints of a curve which may not necessarily be a line, but over the small distances in the subimages we search for lines we have found this fast tracing procedure yields an adequate approximation. The line tracing procedure was used in all of the real time continuous motion runs of our vehicle under vision control.

A combination of three factors enabled us to reduce the image processing time for each image sample to about 2 seconds. First, special image processing hardware in our Grinnell GMR 270 display processor was used for the low-level correlation and convolution. Second, only small subimages (50 by 250 pixels) were searched for road edges by the line finding routines. Third, selection from among the possible set of candidate road edges of the actual road edges was accomplished by simple means (q.v.).

The next step in our plans for development of low-level road-finding vision is to integrate several types of feature detectors in a blackboard data structure. We want to evaluate the success of combining intensity, texture and color edge and region features to find road edges. Earlier we said that we relied on the principle "almost anything works in simple cases". For complicated cases, such as we have encountered in actual outdoor road scenes, we have found that none of the techniques we have tried *always* works. We believe that a combination of techniques will enable us to find road edges reliably in a wide range of situations.

Control

The control procedure translates the visual measurements into vehicle motor commands that, if successful, keep the vehicle moving along the road. We evaluated a half-dozen approaches experimentally with our vehicles and analytically. One approach, servoing to keep the road image centered in the forward field of view, excelled in all the measures, by such a margin that we feel it deserves to be considered a fundamental navigational principle for mobile robots.

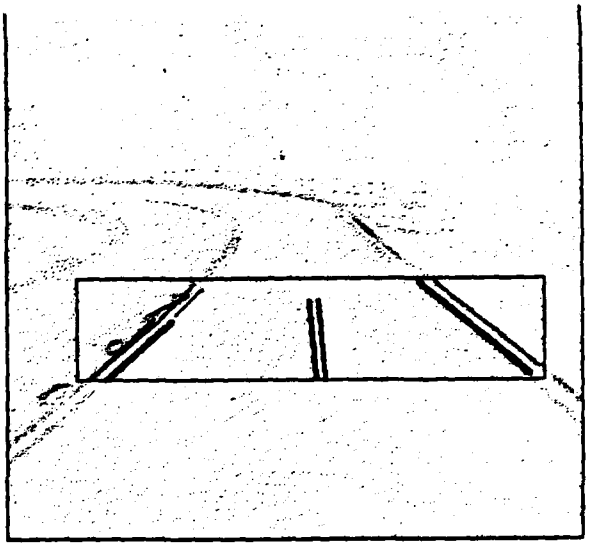


Figure 5: Processing Graphics. Here a road image is shown after processing to enhance intensity changes. The vision program selects a window in which to search for road edges. Candiate left and right road edges are lines fit to the raw edge data, shown here as black lines. Heavy black lines indicate the left and right road edges selected by the program. The computed road center line is shown as a double line.

Let x represent the shortest distance between the center of our vehicle and the centerline of a straight road. θ is the angle between the heading of the robot and the road direction, i.e. when $\theta = 0$ the robot is driving parallel to the road. Suppose the vehicle travels at a constant scalar velocity v, and that control is achieved by superimposing a steering rate, $d\theta / dt$ (where t is time) on top of the forward motion. If there is no slippage, the following kinematic relationship will hold:

(1)

$$dx / dt = -v \sin \theta$$

The general problem for continuous road following is to find a steering function F such that by setting $d\theta / dt = F(x,\theta)$ the vehicle approaches the road center. We tried several functions and noticed a number of recurring problems. Estimating θ and x from the image requires both a precise calibration of the camera and accurate determination of the position and orientation of the road edges in the image. Both are difficult to achieve in practice, and the high noise level in these quantities made most of our functions unstable. A second problem led directly to our solution. The road image sometimes drifted out of the camera's 40 degree field of view, and in the next sampling period the program would fail to find a road, or (worse) identified some other feature, like a door edge, as road. The obvious solution was to servo to keep the road image centered. Experimentally this approach was a stunning success. Besides helping the vision, it seemed to be insensitive to even large calibration errors and misestimates of the road parameters.

The theoretical analysis was remarkably sweet also, and bore out the empirical observations. A first order analysis, where we assume the road image is kept perfectly centered, gives the relation

(2)

$$x / r = \sin \theta$$

where r is the distance in front of the robot where a ray through the camera image center intersects the ground (i.e. the range at which we do our road finding). The parameter r can be changed by raising or lowering the camera, changing its tilt, or by using a different scanline as the center of the region in which road edges are sought.

Equation (2) can be substituted into (1) to give

(3)

$$dx / dt = -v x / r$$

which can be solved directly, giving

(4)

$$x = x_0 e^{-vt/r}$$

where x_0 is the initial value of x when $t = 0$, so to first order the vehicle approaches the centerline of the road exponentially with time.

A more detailed analysis considers the actual servo loop behavior. The displacement of the road centerline image from the center of the forward field of view is proportional to

(5)

$$(\sin \theta - x / r) / \cos \theta$$

Servoing the steering rate on (5) sets

(6)

$$d\theta / dt = -g (\sin \theta - x / r) / \cos \theta$$

where g is the servo loop gain. The full behavior of the robot can be found by solving (1) with (6) simultaneously. These equations are made linear and easily solvable by the substitution $Q = \sin \theta$, giving

(7)

$$dx / dt = -v Q$$
$$dQ / dt = -g (Q - x / r)$$

By co-incidence or cosmic significance of all the servo functions we considered, only this one yielded a fully general analytic solution.

The solution has three cases distinguished by the sign of the expression

(8)

$$gr - 4v$$

In all cases the solution converges to $x = 0$, Q (and θ) $= 0$ exponentially with time. When $g < 4v/r$ the convergence is a decaying oscillation - the sluggish steering causes repeated overshoots of the road center. When $g > 4v/r$ the solution contains a second exponential, and the robot approaches the road center more slowly. When $g = 4v/r$, the critically damped case, we have the fastest convergence and no overshoot, and the behavior is given by the equations

(9)

$$x = e^{-2vt/r} (vt (2x_0/r - Q_0) + x_0)$$

(10)

$$Q = e^{-2vt/r} (2vt/r (2x_0/r - Q_0) + Q_0)$$

The gain sets the turn rate required of the robot. Note that to retain the critically damped situation while increasing v without changing g, it is necessary only to increase r, i.e. arrange to have the vision look further ahead.

The method is successful for several reasons. It keeps the road in view at all times. Because the system always converges, errors in g or camera calibration do not jeopardize performance. Because the parameter being servoed is the most robust direct measurable, namely road position in the image, the noise problems of the other approaches are almost totally eliminated. In particular, θ (or Q) and x though they occupy a central position in the theoretical analysis, need never be calculated in the actual servo loop.

Conclusions

We have developed a vision and control system for a mobile robot capable of driving the vehicle down a road in continuous motion. The system has been tested on two mobile robots, Neptune and the Terregator, in both indoor (hallway and artificial road) and outdoor (asphalt paths in a park and cement sidewalk) environments. In our best run to date the Terregator traversed a 20 meter outdoor path at 2 cm/sec. Image processing time has been reduced to 2 sec/image.

Failure modes of our vehicle have included driving off the road, driving into trees and walls, and driving around in circles. Such failures were mostly due to bugs in our programs, imprecise calibration procedures, and limitations of current hardware (e.g., B&W camera with narrow angle lens), not fundamental limitations of the techniques used.

Future Work

There are several areas that we plan to address. First is the construction of a true **testbed**. This involves mostly software engineering, such as cleaning up and documenting the interfaces between vision and control. This will enable us to try **other vision methods**, such as texture and color operators.

Further work will require the use of a **map**, along with program access to a **magnetic compass** and a **gyro**. The map will list road direction, width, appearance, and intersections, which will provide strong cues to both the image processing and the navigation system. The compass, along with the map information, will help predict road location in the image. This will become increasingly important as we venture onto curved and hilly roads, and as we encounter intersections and changes in the road surface.

The next step is **obstacle avoidance**, which will require limited 3D processing. Projects in the CMU Mobile Robot Laboratory have already demonstrated obstacle avoidance with sonar [2] and stereo cameras [4]; we intend to integrate these into the testbed. Later work may add a laser rangefinder and programs to handle that data.

Finally, as the testbed becomes more complicated, system control will become a major issue. We plan to work on a **blackboard** system with cooperating and competing knowledge sources. All the data, from the lowest level signals to the highest level models and maps, will be on the blackboard and available to all processes.

Acknowledgements

We would like to thank first of all Pat Muir for his work on analysis of the control of the Terregator. Many thanks also to Mike Blackwell, microprocessor hacker *extrodinaire*, Kevin Dowling, tender of robots, and John Bares, prime mover of the Terregator, without whom the experiments described here would have been much more difficult and much less fun. Thanks also to Gregg Podnar and Bob Spies for video and digitization work. Finally, we would like to express our appreciation to Raj Reddy for his support and encouragement.

References

[1] Inigo, R. M., E. S. McVey, B. J. Berger and M. J. Wertz. Machine Vision Applied to Vehicle Guidance. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(6), November, 1984.

[2] H. Moravec and A. Elfes. High Resolution Maps From Wide Angle Sonar. In *IEEE Conference on Robotics and Automation*. 1985.

[3] Rosenfeld, A. and A. C. Kak. *Digital Picture Processing (Vol 1)*. Academic Press, 1976.

[4] C. Thorpe. *FIDO: Vision and Navigation for a Mobile Robot*. PhD thesis, Carnegie-Mellon University, 1984.

[5] Wallace, R. S. A Modified Hough Transform for Lines. In *Computer Vision and Pattern Recognition*. IEEE, 1985.

[6] Waxman, A. M., J. LeMoigne and B. Scrinivasan. Visual Navigation of Roadways. In *International Conference on Robotics and Automation*. IEEE, 1985.

[7] W. Whittaker. *Terregator - Terrestrial Navigator*. Technical Report, Carnegie-Mellon Robotics Institute, 1984.

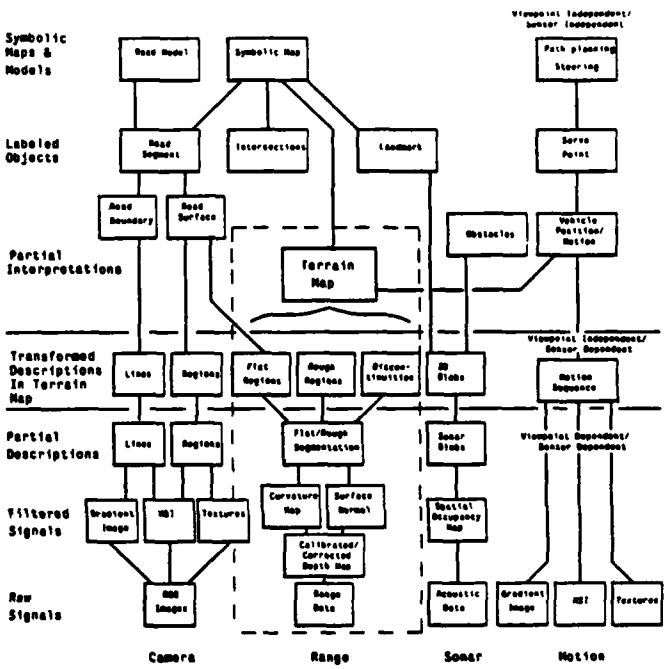


Figure 6: Blackboard System. We have begun the design of a blackboard system to integrate the multiple sensors, knowledge sources and vehicle actuators planned for the CMU ALV system. Our future work is to embed the modules we already have in the blackboard system, with multiple parallel knowledge sources accessing a global data base, and also to add other modules.

A Modified Hough Transform for Lines

Richard S. Wallace

Department of Computer Science
Carnegie-Mellon Univeristy
Pittsburgh, PA 15213

Abstract

A new parameterization for lines in images is presented with application to the Hough transform. The modified Hough (Muff) transform has several implementational advanatages over the conventional $\rho-\theta$ parameterization. The Muff transform parameter space is better suited to computer graphics line drawing routines. The Muff transform requires no transcendental function calls or table lookup. The relation between the tesselation of the parameter space and the resolution of the lines represented is discussed. The shape of the Muff space is amenable to compaction into a rectangular array. The implementation of the Muff transform is presented.

The Hough transform can be used to find lines in images[1,2]. Each edge element in the picture votes for all of the lines that could possibly pass through it. The voting takes place in a two-dimensional parameter space, where each line is represented as a point. This space is tesselated into a grid of rectangular cells, and each cell accumulates votes for lines represented by values in that cell. In implementations of the Hough transform the tesselated parameter space is an accumulator array. To extract the lines in an image, the Hough parameter space is searched for peaks which lie above some thresshhold: these are assumed to correspond with lines in the image. Two problems which arise in the implementation of the digital Hough transform are the selection of the Hough parameters and the choice of granularity of tesselation for the parameter space. The usual parameters selected to represent lines in Hough space are ρ and θ, where lines are given by the expression $\rho = x\cos\theta + y\sin\theta$. These parameters have the advantage over m and b in the $y = mx + b$ form that they are bounded. It is easy to see that for an rectangular image extending from (x_{min}, y_{min}) to (x_{max}, y_{max}) the values of ρ and θ are bounded by $-\sqrt{x_{min}^2 + y_{min}^2} < \rho < \sqrt{x_{max}^2 + y_{max}^2}$ and $0 < \theta < \pi$, whereas m and b are unbounded. This paper presents a different bounded parameterization of lines in an image and several advantages of this new representation over the $\rho - \theta$ parameters.

The new Hough line parameterization is illustrated in figure 1. We assume for simplicity that the image is bounded by a rectangle parallel to the x- and y-axes and extending from the origin to some (x_{max}, y_{max}). A bounding box extending around the image provides the basis for the parameterization. A line passing through the image is parameterized by the two points where the line intersects the perimeter of the bounding box. These points are given by their distance along the perimeter of the bounding box, where distance is measured counterclockwise along the box starting at the origin. Thus a line has two parameters, s_1 and s_2, representing the two points where the line intersects the box. To preserve uniqueness of the representation, we assume $s_1 < s_2$. The range of possible value are $0 \le s_1 < s_2 < 2(x_{max} + y_{max})$. This new parameterization is called the Muff transform.

An immediate advantage of the Muff transform is purely graphical. The transform parameters easily map back into points on the image's bounding rectangle. The line represented by (s_1, s_2) in figure 1, for example, passes through the image at $(s_1, 0)$ and $(2x_{max} + y_{max} - s_2, y_{max})$. These points can be passed directly to a computer graphics routine to draw the line. No clipping is needed. The calculation of the endpoints for a line $\rho - \theta$ requires more work. First, the peak value indecies in the accumulator array must be mapped back into their corresponding $\rho - \theta$ values. These are then used to write a line equation of the form $Ax + By + C = 0$, which then must be solved for x and y at each side of the rectangle. The Muff representation requires at most two subtractions to determine both endpoints.

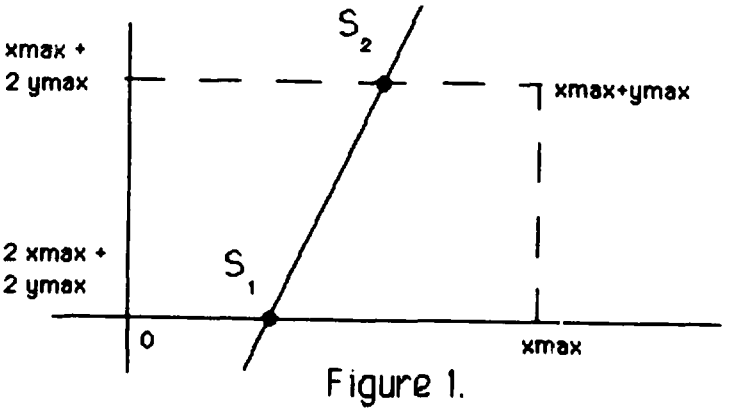


Figure 1.

CH2145-1/85/0000/0665$01.00©1985 IEEE

Implementations of the Hough transform for lines can suffer from two problems related to the use global edge information in the image to find local lines. First, edge elements on colinear but not connected line segments vote for the same line. Second, the transform of an image with many noisy edge points or many irregular blobs may contain spurious lines linking distant edge elements because the thresshhold on peak detection in the Hough space must be set low in order to obtain any lines. The pure Hough transform does not preserve information about which edge points voted for a particular line and hence the transform cannot find directly the endpoints of line segments. One obvious solution is to store in each bin of the Hough space not only the count of edge elements voting for a particular line, but also a list of the pixel coordinates of the edge elements themselves. Later processing can then fit line segments to connected sets of pixels in peak Hough bins. Another approach is to divide the image into a number of smaller rectangular regions, and compute the transform for each. The Muff transform is best suited to the latter.

The transform is implemented by the following procedure. Given an edge element (a, b) find the point (c, d) on the bounding box so that $(0, 0)$, (a, b) and (c, d) are colinear. (c, d) is parameterized by a value s_{max}. Then for each s_1, $0 < s_1 < s_{max}$, and the point along the box associated with s_1, there is another point given by s_2 so that s_1's point, (a, b) and s_2's point are colinear. The calculation of s_2 is straightforward and depends on which side of the rectangle a line intersects. In any case the computation of s_2 from s_1 and (a, b) reduces to the problem of intersecting a line in two-point form with a horizontal or vertical line[3]. Thus an advantage of the Muff transform over the $\rho - \theta$ transform is that no transcendental function calls are needed. The need for actual transcendental function calls can be eliminated in implementations of the $\rho - \theta$ transform algorithm by table lookup, however. The $\rho - \theta$ transform requires only divisions and no transcendental function calls or table lookup.

The choice of tesseslation for the parameter space affects the resoltion of the lines which can be found. Intuitively, the finer the tesselation, the finer the accuracy of the lines which can be represented. One measure of resolution is the distance measured between two lines where they intersect one side of the image. For the x-axis, let's call this distance Δx. Figure 2 shows that for the $\rho - \theta$ representation the resolution Δx is a function of ρ. The further the line from the origin, the coarser the representation. In the Muff representation, however, the resolution Δx is constant around the perimeter of the image. In all fairness, the angular resolution of lines in the Muff representation is finer near the corners than near the center of the image. The Muff representation, however, caputures exactly the set of lines that can be drawn by computer graphics from one point on the rectangle to another, up to the resolution of the tesselation. The absolute upper bound on the useful size of the Muff parameter space is $(x_{max} + y_{max})^2$, where x_{max} and y_{max} are respectively the number of pixels along the x-axis and y-axis sides of the image, because no more lines than this number can be drawn by graphics from a pixel on one side to a pixel on another side of the image. The $\rho - \theta$ representation will not represent this entire set or represent some of its elements redundantly, depending on the granularity of the parameter space tesselation.

A peculiarity of the $\rho - \theta$ form for the Hough space is that, although it is bounded, it has an irregular shape (see fig. 3). Not all pairs of (ρ, θ) in the rectangle given by $-\sqrt{x_{min}^2 + y_{min}^2} < \rho < \sqrt{x_{max}^2 + y_{max}^2}$ and $-\pi/2 < \theta < \pi$ represent possible lines in the image. If minimizing storage were an issue in a Hough transform implementation, the compaction of the $\rho - \theta$ space would prove difficult. Figure 3 shows the set of possible values in the Muff representation. It is clear how these could be compacted into a rectangular array if necessary. Also, it is simple to write an algorithm which efficiently scans only the Muff array's possible cells for peaks or local maxima. For each row in the accumulator array, the cells can be scanned from left to right starting at the first possible value in that row.

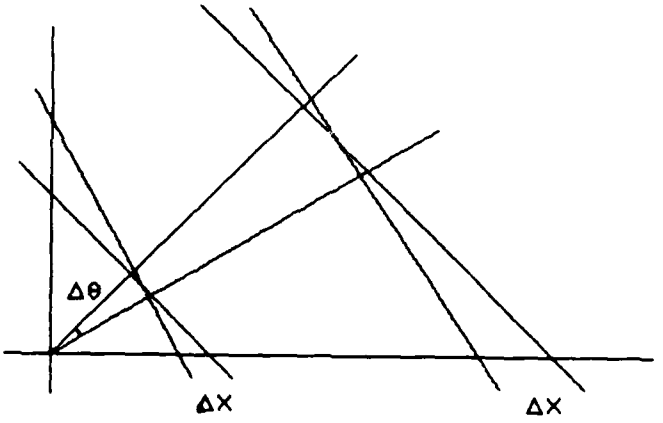
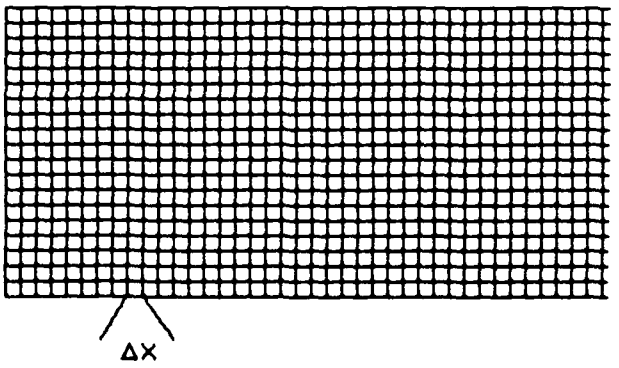


Figure 2. In the ρ-θ representation the resolution of lines that can be represented is a function of ρ and θ. This diagram illustrates two lines that appear adjacent in the tesselated transform space. The distance between these lines where they intersect the x-axis grows as ρ increases.



In the Muff representation the resolution of lines represented is constant throughout the space. The muff representation captures exacity the set of lines that can be drawn across the rectangle by computer graphics.

The Muff transform has been implemented and tested on images of roads. In the road following application, it is not usually necessary to find the endpoints of line segments in the image. It is the road edges which are important, and these can be assumed to extend from one side of the image to another. Thus neither the technique of storing pixel locations in the Hough accumulator array nor the method of dividing the image up into smaller rectangles is used. Road edges tend to be strong and extend over the whole image, so the Muff transform picks them out easily. The design of special purpose voting hardware[4] has made the use of the Muff transform more practicable for real-time vision tasks.

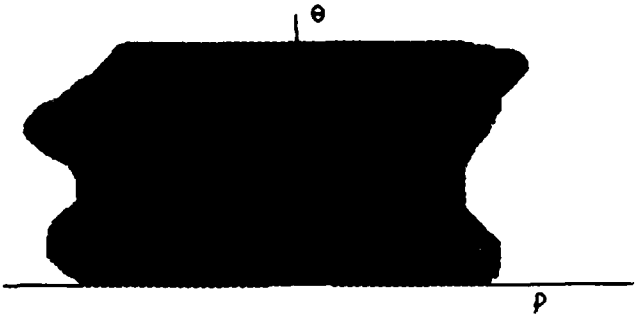
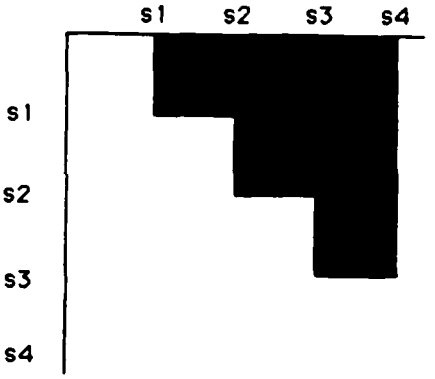


Figure 3. The set of possible values of ρ and θ for lines passing through a rectangular image defines an irregular shape in the ρ-θ plane. The irregularity makes it difficult to compact the useful values into a rectangular array.



The set of plausible pairs of parameters in the Muff space can be easily compacted into a rectangular array. In this diagram, s1 = xmax s2 = xmax+ymax, s3 = 2xmax+ymax, s4 = s(xmax+ymax).

[1] Duda, Richard O. and Peter E. Hart "Use of the Hough Transform to Detect Lines and Curves in Pictures" CACM vol. 15 no. 1, January, 1972. pp. 11-15.

[2] Ballard, Dana H. "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition* vol. 13 no. 2, 1981. pp. 111-122.

[3] Bowyer, Adrian and John Woodwark *A Programmer's Geometry*, Butterworths, 1982.

[4] Sher, David and Tevanian, Avidas "The Vote Tallying Chip: A Custom Integrated Circuit", Custom VLSI Conference, Rochester, May, 1984.

Progress in Robot Road-Following

R. Wallace, K. Matsuzaki, Y. Goto,
J. Crisman, J. Webb, T. Kanade

Robotics Institute, Carnegie-Mellon University

Abstract

We report progress in visual road following by autonomous robot vehicles. We present results and work in progress in the areas of system architecture, image rectification and camera calibration, oriented edge tracking, color classification and road-region segmentation, extracting geometric structure, and the use of a map. In test runs of an outdoor robot vehicle, the Terregator, under control of the Warp computer, we have demonstrated continuous motion vision-guided road-following at speeds up to 1.08 km/hour with image processing and steering servo loop times of 3 sec.

1. Introduction

Research in robot navigation on roads is part of the Autonomous Land Vehicle Project (ALV) at Carnegie-Mellon University. Broadly, our work is aimed at creating autonomous mobile robots capable of operating in unstructured environments. To this end, our research program involves a variety of sensors, programs and experimental robot vehicles. This paper is focused on recent progress in detection of and navigation on roads, using a TV camera as our sensor and a six-wheeled outdoor autonomous robot, the Terregator [7], as our test vehicle. We present results and work in progress in the areas of system architecture, image rectification and camera calibration, oriented edge tracking, color classification and road-region segmentation, extracting geometric structure, and the use of a map.

For robot navigation of roads, we use a single television camera as our primary sensor. In this application, the monocular TV camera is considered superior to ranging sensors such as laser scanners or sonar for three reasons. First, roads we are interested in following do not necessarily have prominent 3-dimensional features at their shoulders; most often there is no depth discontinuity between the road surface and the surrounding roadside. Second, we have developed one steering strategy that servos the vehicle based on measurements in the image plane itself, rather than on measurements in a world coordinate frame. Third, we have so far relied on a *local ground plane* assumption, that the ground around the vehicle is locally planar, so that any time we do need to transform image points to world coordinates, the transformation is trivial.

To attain the broad goals of our project, we have split the research into two efforts. The goal of the first effort is to develop a road-following system which uses a map to navigate around a highly structured and visually simple network of sidewalks on the CMU campus. The goal of the second effort is to develop vision routines for road-following in a less structured and visually more complex environment in a nearby park.

2. Sidewalk Navigation

The sidewalk environment at CMU is a network of mostly straight concrete pathways joined at intersections of various shape. The sidewalks have fairly uniform color and texture and are always surrounded by well-groomed grass, giving them consistent high-contrast edges. The goal of our research in this environment is to develop algorithms for geometric reasoning, shape-matching and navigation with a map.

2.1 Map and Blackboard

The overall system architecture to which a vision-based road-following subsystem interfaces is a *blackboard* [5], a shared *memory structure containing a local map of the robot's* environment. Other sensing processes, such as those interpreting range data, and other knowledge-based processes, such as those updating the local map, are also tied to the blackboard.

2.1.1 Dialogue Model

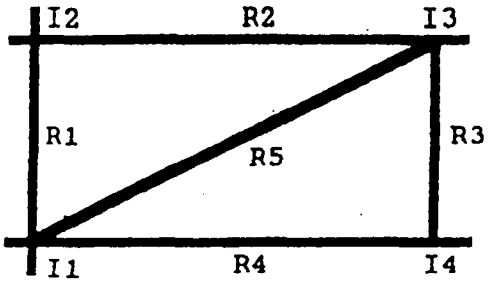
The road-following subsystem consists of four modules; Vision, Map, Navigator, and Motion Control. These modules communicate with each other by sending and receiving tokens through the Blackboard. In selecting this decomposition of our system into modules, we followed the principle of *information hiding*. The Vision module contains expertise needed for extracting features from images. The Map module knows the structure of the robot's environment and its position. The Navigator is responsible for planning paths. The Motion Control module insures that the vehicle executes navigation commands. Thus each module has a different domain of expertise. For example the Vision module does not know the robot's map or route. That information is kept hidden and is used only by the Map module to make predictions to the Vision module.

Communication between the various modules looks like a dialogue. Figure 1 shows the dialogue model of the road-following subsystem. This model reflects the information hiding principle of the design. In the example, the Map hides information from the vision module, except for the facts which are relevant for the current scene. The Map tells the Vision module only about the predictions it makes for the current scene.

With map data, the Map module produces the token, Predicted Object, which shows what the Vision system shall see. For example, a Predicted Object can be a road or an intersection. Using Predicted Object, Vision sees and makes the token, Detected Object, which shows the shapes of objects in front of the vehicle. Using Detected Object, the Map decides the vehicle's Current

[1]Currently, this project is funded in part by Carnegie-Mellon University, by the Office of Naval Research under contract number N00014-81-K-0503, by the Western Pennsylvania Advanced Technology Center, by Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539, and by Denning Mobile Robotics, Inc. Richard Wallace thanks NASA for supporting him with a NASA Graduate Student Researchers Program Fellowship Grant.

Position. Using Current Position and the map data, the Navigator supplies the token, Motion Command, which tells how to drive the vehicle. Using Motion Command, the Motion Control drives the vehicle.



User: Robot is at road R1, 3 meters from I1.
Navigate to R3, 2 meters from I3.
Map: Vision will see straight road and cross-type intersection.
The color in the left is ... Detect them.
Vision: Ok. I found them. Their shapes are ...
Navigator: Drive on it 2.5 meters and turn to right 90 degrees.
Motion Contol: Ok. I drive. (vehicle moves)
Map: Vision will see straight road. The color on the left is ...
Detect it.

Figure 1: Dialogue Model of Map Interface

In the road-following subsystem, two kinds of coordinate systems, World Coordinate and Vehicle Coordinate, are used. World Coordinate is an absolute coordinate. The map data is written with World Coordinate. The Vehicle Coordinate frame, which is fixed on the vehicle, is used by Vision to represent Detected Object, because it does not know where the vehicle is. Coordinate transformation is done when necessary.

2.1.2 Predictions

The map module supplies *predictions* to the vision module. The map data consists of two kinds of maps, a topological map and a geometrical map. The topological map stores the topology of roads and intersections. The geometrical map stores the shapes of roads and intersections.

With these map data, the Map predicts the kinds, the shape and the image features of objects which shall be seen in a camera view. The purpose of detecting objects is to navigate the vehicle. The detail of an object shape is trivial and therefore, not necessary for navigation. The Map creates *interest segments*, which are the primary edge line segments of roads and intersections. The interest segments are enough for Map to decide the vehicle's Current Position and the object shape necessary for navigation. They are likely to be the edge segments most easily detected by Vision, and therefore are included in the Predicted Object. An interest segment is also a key for matching. We discuss this in detail below.

2.2 Extracting Geometric Structure

Our Autonomous Land Vehicle has to not only follow single road, but also to detect an intersection and turn into one of the intersecting roads. In this case accurate shape of roads and an intersection has to be extracted. This is difficult because variations in camera view and imaging conditions result in variations in the shapes detected. Furthermore there are many factors which make it difficult to detect a road shape such as cracks, dust, gaps between concrete slabs. They are not noise but physical substance, therefore even if region classification is done perfectly, they possibly remain. To solve these problems, we implemented two procedures. First, the image is processed to eliminate these disturbing factors and to reproduce the road region. After that, using knowledge from map, interest segments, which are key to decide an position of an intersection, are found.

2.2.1 Reproducing the Road Region

To eliminate the disturbing factors, two phase image processing is done; extracting high-confidence road regions and then connecting them.

The result of region segmentation includes four types of segments: 1)actually road and classified as road, 2)actually not road and classified as not road, 3)actually road but classified as not road, 4)actually not road but classified as road. At the first image processing phase, the program selects a conservative classification threshold so that only ideal road surface is classified as road. This result includes much type 3 region but little type 4 region, and region classified as road is confidently road. Then, to cover type 3 region, we did a combination of reducing resolution and expansion/contraction of image.

The expansion/contraction method is known as a good method to eliminate gaps or small holes, but calculation time is long when the size of defects are large and large number of expansion/contraction is needed. We have to use this method in real time during vehicle running. So, we reduced resolution before expansion/contraction. This method absorbs several pixels into one pixel, and decides the the new pixel value by a threshold on the proportion of original pixels classified as road to nonroad. We use a reduction ratio of 8*8 to 1 pixel followed by 1 or 2 iterations of expansion/contraction. This obtained both sufficient shape estimates and quick calculation.

2.2.2 Polygon Fitting

To recognize an intersection from the reproduced shape, we fit a polygon to the intersection contour. Shape analysis based on polygon is much quicker than one based on whole pixels or run-length data. The processing includes following steps.

1. **Extracting Straight Line.** Most of roads imaged are straight but if they include curves, these can be represented as a set of segmented straight lines. So, we apply a polygonal approximation to original precise polygon to extract major straight components. The tolerance is set so that the interest segments can be picked up well.

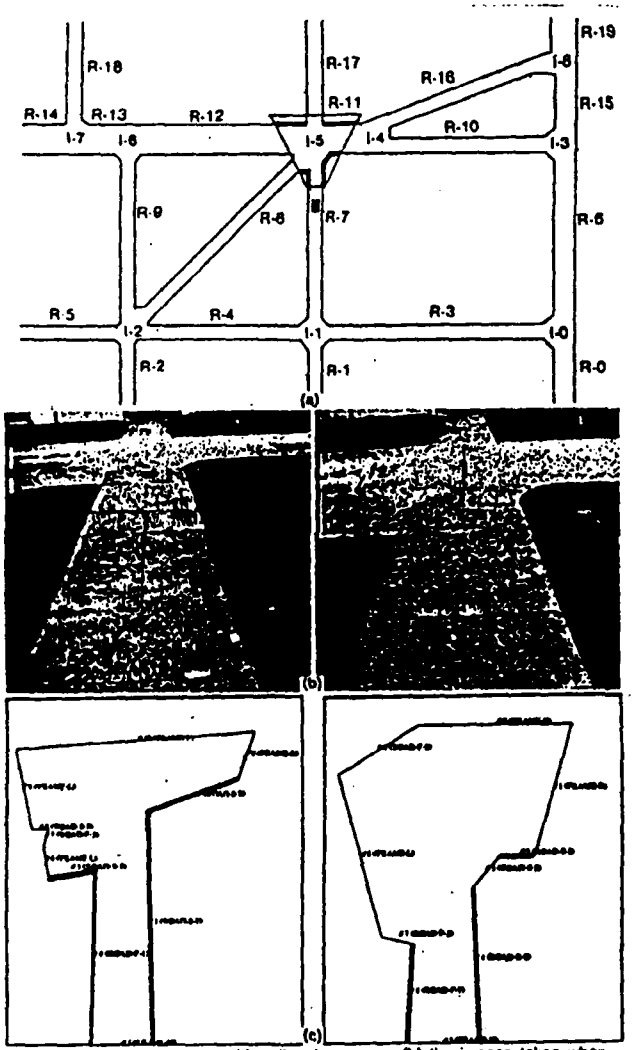
2. **Labeling Lines.** We have developed a program which labels lines. At first,this program identifies viewing frame edge lines by searching lines which are contained in the Coordinate of viewing frame. Second, this program classifies lines by angle and gives same labels for the similar angle lines. The Map module produces also the description of interest segments which shows the segment attribute and the relationship between segments. Using this description, this program can match the classified lines to the predicted interest segments easily. The list showing the detected segments and their correspondence to the predicted is returned to the Map module. Understanding of whole geometric structure is done by the Map in next map matching step.

2.2.3 Map matching

With the result of the Vision module and the object prediction, the Map module can know the names and the shapes of the detected objects. In order to estimate the vehicle current position, the Map module selects crossing lines in the detected objects and corresponding lines in the map data, and calculates coordinates transformation which can match them. In this stage, when only

straight portion of the road is in the view frame, the measurement from the Vision module can constrain the vehicle position and orientation only perpendicular to the road. In such case, the location along the road is calculated using the vehicle motion. The positional error which might accumulate along the path will be corrected as the vehicle approaches to the intersection and can see the road edges in multiple orientations.

Figure 2 shows a result of CMU campus sidewalk run. Along the vehicle approaches an intersection, the vision module detects different parts of road contour which are predicted as major line segments by the map module.



Navigation on the campus sidewalk using a map: (b) the images taken when approaching intersection I-5. The trapezoidal region in (a) represents the predicted view of the Vision. (c) the results of road region extraction of the images in (b). The images are rectified into the map coordinates from the image coordinates. The edges matched with prediction are indicated by bold lines.

Figure 2: Navigation on Campus Sidewalk using Map

3. Park Road Following

Our park environment contains a 1 kilometer curving asphalt path part of which is always illuminated directly and part of which is shaded by trees. The path itself varies in texture from mostly smooth and featureless to cracked and pot-holed, and in color from blue-gray to black. The shoulder around the path consists mostly of grass, but there are also some sections of dirt and rock. Seasonally, both road and shoulder are obscured by leaves, snow or ice. Trees and their shadows are also present. The main goal of our research in the park environment is to develop vision algorithms capable of steering the vehicle reliably in this unstructured environment.

3.1 Road-Edge Following

We have developed a technique for tracing the edges of a road using an oriented edge detector. Like the tracker discussed in [9] our algorithm begins with an estimate of the start position from which is the edge is to be traced. Unlike that tracker, ours integrates or smooths the edge along the edge direction. Integrating the signal along the direction of the edge has the effect of smoothing and reducing noise content. Then, the position of the edge is localized by matching an ideal step edge model with the one-dimensional cross-section.

Oriented edge detection operators have been explored in computer vision, with perhaps the best results found in [2]. We chose an oriented operator since it is more reliable than an unoriented one. For example, if the road in the image is oriented at 45 degrees, then a conventional edge detector will find gradually sloping intensity values, see figure 3. However, if the same detector is oriented at 45 degrees, then the oriented detector would see a sharp change in intensities, and therefore, the edge location is detectable. We have implemented edge operators at a number of different orientations so that we can obtain a reliable response regardless of the orientation of the road in the image.

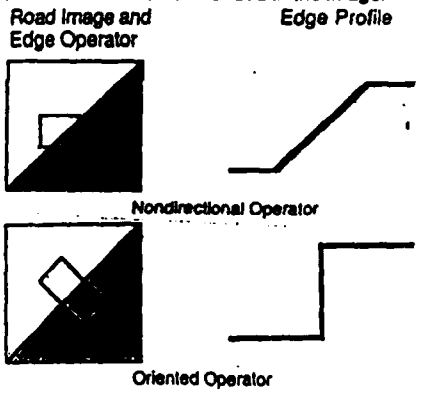


Figure 3: An Oriented Edge Operator

3.2 Implementation

The edge tracer constructs a list of road edge points in an image given a position (r_0, c_0) and orientation, θ_0 of a road edge. The oriented edge operator integrates the signal along its columns. If the operator does not align with the image columns, then it selects pixel values nearest to the position of its columns for the summation. This one dimensional result of the edge operator is called the *edge signature* or *edge profile*.

Then a new road edge point, (r_p, c_p), is predicted to lie a distance from (r_0, c_0) at an angle of θ. A search window is created centered at (r_p, c_p), oriented at the angle θ. The edge operator creates an edge profile in the search window. The road edge, (r_i, c_i), is determined to be where the an ideal step edge and the window profile have the best correspondence. The orientation of the road is recalculated by $\theta = \arctan2(c_i - c_{i-1}, r_i - r_i)$. This algorithm is iterative if $(r_i, c_i) \rightarrow (r_{i-1}, c_{i-1})$. This process is repeated until the search window falls outside of the image bounds.

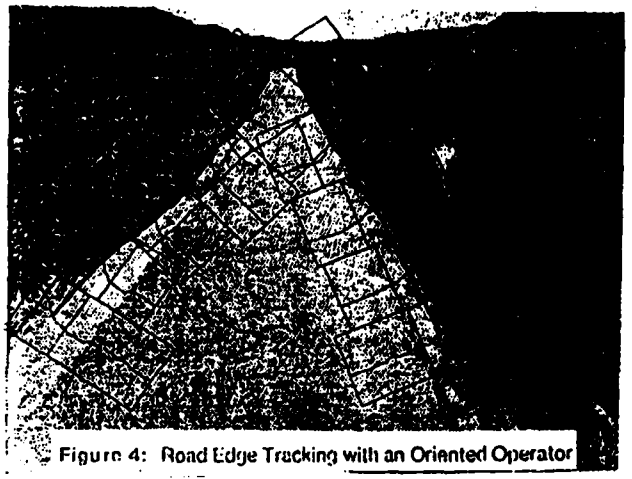


Figure 4: Road Edge Tracking with an Oriented Operator

3.2.1 Results

The edge tracer has been tested on 480 X 512 grey level images. The dimensions used for the search window were 64 rows by 128 columns. Figure 4 shows a typical result of the edge tracor. The initial position is given near the bottom of the image and the oriented edge detector proceeds upward in the image. The larger boxes outline the search windows, and the smaller, inner boxes show the positions of best correlation. The edge profiles are shown inside the search windows.

We have developed a vehicle driver system based on oriented edge tracing. The initial position and orientation of the left and right road edges are input to the system and used for the first iteration of the oriented edge tracer. After finding the road edges in the image, they are back-projected to the ground plane. The vehicle motion between images is used to locate the previously found road edges relative to the vehicle. Then the previous edges are projected in the new image. These edge locations are used for the position and orientation estimations required for the edge tracer. The 3D projection of the road edges also allow the right and left road edges to be tested for parallelism and proper separation.

This system works well on images where there is a fair amount of contrast between the road edge and the road shoulder. We have been able to drive our vehicle quite reliably on gently curving roads. However, we have had difficulty when the edge of the road lies close to obstacles or when shadows lie on the road. The edge tracer can locate a road edge point in under one second. The system can drive the vehicle at speeds up to 0.3 meters/sec.

We are currently working on testing the road edges found by the edge tracer for geometrical consistency. If the right and left edges of the road are not parallel and the proper width apart, then the system must decide which edge should be used to drive the vehicle. Measures of evaluation based on the height, width, smoothness, and consistency are currently being tested. If these measures are reliable, the system should be able to evaluate its performance .

3.3 Road-Region Segmentation

The second major approach to road feature detection is region segmentation. This differs from the edge-based procedure in that the road itself is extracted, rather than its contours. As we mentioned earlier, the edge information can be used to verify and localize the region hypothesis. Region classification is based on assignment of region labels to all pixels in an image, where the assignment depends on properties of that pixel such as brightness, texture and color around that pixel. Our work is focused on color classification.

3.4 Color

Early in our work on visual detection of roads we recognized the importance of utilizing color vision sensors. We found in black-and-white images of our test site that the perceived intensity of the asphalt road differed very little from the intensity of the surrounding grass, although the color was very different. Gray-level histograms of the images were either very flat, or they had peaks caused by shadows and highlights, rather than road or nonroad features. Histogram-based segmentation techniques and edge operators failed for the same reason. We considered texture energy measures to segment road and grass, since the grass has more edges per unit area, but the noise introduced into the images by an inferior TV transmission system confounded attempts to measure high-frequency texture information. Even in the presence of high spatial frequency image noise color information is retained.

3.4.1 Pixel Classification

In color images each pixel (x, y) has an associated color vector $(R(x, y), G(x, y), B(x, y))$. The set of all possible (R,G,B) values forms a color cube RGB. The RGB cube can be divided in various ways so that pixels having certain color vector values can be classified as road or nonroad. A simple region classification involves selecting a sample road region and grass region from a training image, and using the average values $(\mu R_{road}, \mu G_{road}, \mu B_{road})$ and $(\mu R_{grass}, \mu G_{grass}, \mu B_{grass})$ as ideal feature points in RGB space. If the covariance matrices Σ_{road} and Σ_{grass} are also measured then the colors can be modeled as trivariate normal distributions (TVNDs). The result of a TVND model is to divide color space into regions separated by quadratic surfaces. Figure 5 shows a result of classifying a sequence of rectified road images from the park site.

3.4.2 Color variation

Unfortunately the color of road and shoulder do not remain constant from one image to the next. Variation in color arises for a variety of reasons, such as illumination changes (e.g. shadow versus direct illumination) and material changes (e.g. dry asphalt versus wet, green grass versus yellow). Additionally, our test vehicle is equipped with a TV broadcast station, through which images are transmitted to a fixed-based computer. The chromatic component of the TV signal varies depending on such factors as the position of the robot vehicle with respect to the TV receiver.

We have begun to explore the use of adaptive color models to reduce the problems arising from color variation.

3.4.3 Shadows and normalized color

Shadows cause many of the failures of our vision system. Edge-based schemes for detecting road edges are fooled by high-contrast shadow edges, as shadow edges often have a greater brightness-to-darkness ratio than material edges. Even region classification schemes based on color are confounded by shadows

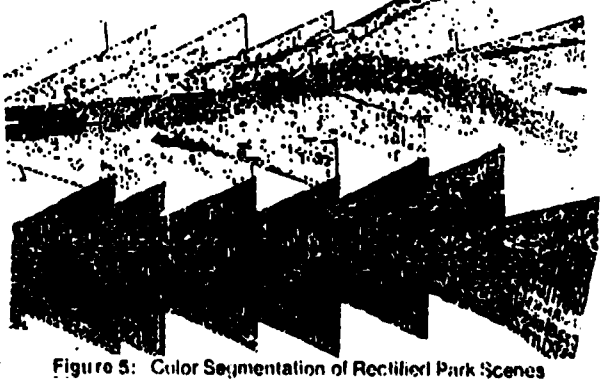


Figure 5: Color Segmentation of Rectified Park Scenes

because images of objects in shadow contain color values clustered around different points in *RGB* space.

Consider an object imaged with color C_1 in a sunlit part of the scene and color C_2 in a shadowed region. To a first approximation, $C_1 = kC_2$ for some constant k. This is because the object reflects the same color in shadow, it is just imaged at a different intensity. Thus a preprocessing step is to *normalize* all the color vectors of an image, by transforming each point $(R(x, y), G(x, y), B(x, y))$ into $(r(x, y), g(x, y), b(x, y))$ such that

$$r = R/(R+G+B), \quad g = G/(R+G+B), \quad b = B/(R+G+B).$$

Then all the color points lie on the plane $R + G + B = 1$.

Although the transformation from *RGB* to *rgb* is sufficient for erasing shadows in many cases, it is not always successful. There are two factors limiting its usefulness. First, the dynamic range of a TV camera is not very large (a maximum brightness:darkness ratio of 7:1) compared with film (a maximum brightness:darkness ratio of 20:1) or the human eye (a maximum brightness:darkness ratio of at least 1000:1). Thus TV images containing of shadowed regions may have splotches of maximum bright or dark, in which all spatial detail and color information is lost. Color normalization will not work in these areas. The second factor is less important, but easier to work around. Nonshadow areas in our outdoor road scenes are illuminated by direct sunlight, which has a more-or-less constant spectral distribution. Shadowed regions are illuminated by skylight and by sunlight reflected off surrounding objects (such as tree leaves and tree trunks in our case). Thus the reflected color of a shadowed part of a region is not quite the same as the color reflected from that part of the region in direct sunlight. In practice the difference is small enough not to matter for our classification techniques.

Color normalization reduces the dimensionality of color classification to two, in which case a bivariate normal distribution is used as a color feature model.

3.5 Image Rectification

We have implemented programs for nonlinear warping of an perspective of a road to transform it into a view like what we would see if we were flying over the road and looking down on it. This transformation, called *image rectification*, produces a map-like image in which the structure of the road is made explicit. The result is an image which is in vehicle coordinates and can be used for camera calibration, debugging of ground-plane operations, detection of ground-plane features, and display of planned robot paths.

3.5.1 Definition

Figure 6 shows the process of image rectification. It is most easily described by considering a rectangular grid projected onto the ground plane. Grid points can be considered as pixels of the rectified image. Rectification consists of back-projecting the grid-points in the ground plane to the original image, in order to see what intensity value should be placed at that point. Once the back-projection is computed, it is stored as a lookup table so that subsequent images can be rectified quickly.

Figure 7 shows the process of image rectification for a wide-angle fish-eye lens. This lens is superior to a standard reflex lens (which we usually model as a pin-hole) for imaging the road, because the road always remains in view even when the vehicle makes sharp turns off the centerline. The point $(-1, I_A/J_A)$ on the ground plane is first projected onto the unit sphere centered at the origin, then perpendicularly to the image plane which is tangent to the sphere at $(0, 0, 1)$. The overall transformation is

$$(I_C/J_C) = (-1, I_A)/\sqrt{1+I_A^2+J_A^2}$$

where A is the rectified image and C is the original image.

This transformation is more useful if it can be done quickly: we anticipate carrying out this transformation on the CMU Warp

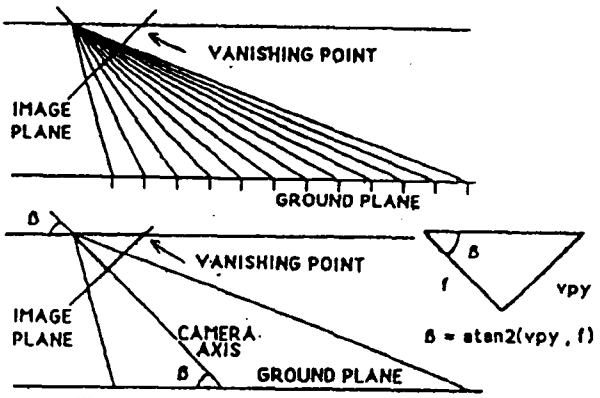


Figure 6: Image Rectification for Pin-Hole Lens and Determination of Camera Tilt

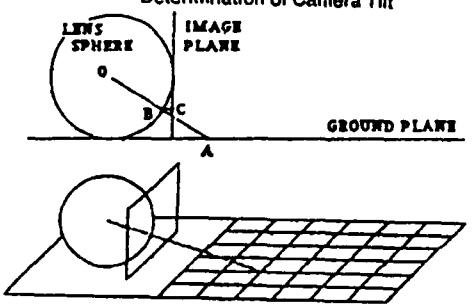


Figure 7: Image Rectification for Fish-Eye Lens

3.5.2 Camera calibration

The image rectification process (for the pin-hole lens model) can be used for camera calibration. By "camera calibration" we mean deriving the necessary parameters for transforming image points to the local ground plane around the vehicle. By intersecting a pair of lines in the ground plane around the vehicle a point on the horizon (vanishing line) can be detected. Note that the *actual* horizon need not be in view, only a pair of lines in the local ground plane. In fact, the lines need only lie in any plane parallel to the ground plane, except the planes containing the camera axis. In practice we use a pair of forward-pointing straight metal poles bolted to the side of the Terregator as a calibration "hood ornament". We hand-select these points from a calibration image.

Once the horizon line is known, the tilt of the camera is easily derived as in figure 6 Given the tilt β of the camera and an estimate of the camera focal length f, the transformation from ground plane points to image points is obtained directly as in figure 6.

A second aspect of camera calibration is determining the x and y scale factors for the image, where x indicates distance along an axis parallel to the vehicle forward direction and y is distance along an axis parallel to the wheel rotation axes. To measure these parameters, we place meter sticks on the ground plane in camera view, digitize and rectify a test image, and then measure the lengths of the meter sticks along the x and y dimensions.

3.6 Warp Runs

In test runs of an outdoor robot vehicle, the Terregator, under control of the Warp computer, we have demonstrated continuous motion vision-guided road-following at speeds up to 1.08 km/hour with image processing and steering servo loop times of 3 sec.

3.6.1 Warp Hardware Description

The Warp machine has three components: the Warp processor array, or simply Warp, the interface unit, and the host, as depicted in Figure 8. We describe this machine only briefly here; more detail is available separately [1]. The Warp processor array performs the bulk of the computation; in this case, low-level vision routines [2]. The interface unit handles the input/output between the array and the host. The host has two functions: carrying out high-level application routines and supplying data to the Warp processor array.

The Warp processor array is a programmable, one-dimensional systolic array, in which all cells are replicas of each other. Data flow through the array on two data paths (X and Y), while addresses and systolic control signals travel on the Adr path (as shown in the Figure 8). The Warp cells are specialized for floating-point operations. The data path of a Warp cell is depicted in Figure 9. Each cell contains two floating-point processors: one multiplier and one ALU [8]. These are highly pipelined; they each can deliver up to 5 MFLOPS each. This performance translates to a peak processing rate of 10 MFLOPS per cell or 100 MFLOPS for a 10-cell processor array. To ensure that data can be supplied at the rate they are consumed, an operand buffer is dedicated to each of the arithmetic units, and a crossbar is used to support high intra-cell bandwidth. Each input path has a queue to buffer input data. A 4K-word memory is provided for resident and temporary data storage.

As address patterns are typically data-independent and common to all the cells, full address generation capability is factored out from the cell architecture and provided in the interface unit. Addresses are generated by the interface unit and propagated from cell to cell (together with the control signals). In addition to generating addresses, the interface unit passes data and results between the host and the Warp array, possibly performing some data conversion in the process.

The host is a general purpose computer. It is responsible for high-level application routines as well as coordinating all the peripherals, which might include other devices such as the digitizer and graphics displays. The host has a large memory in which images are stored. These images are fed through Warp by the host, and result images from Warp are stored back into memory by the host. This arrangement is flexible. It allows the host to do tasks not suited to Warp, including low-level tasks, such as initializing an array to zero, as well as higher level tasks, such as processing a histogram to determine a threshold.

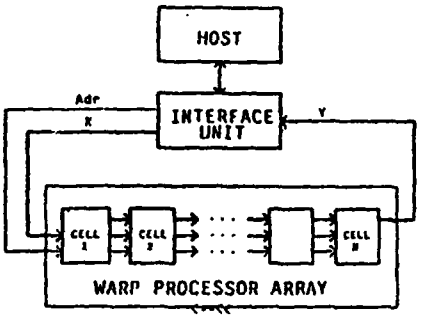


Figure 8: Warp machine overview

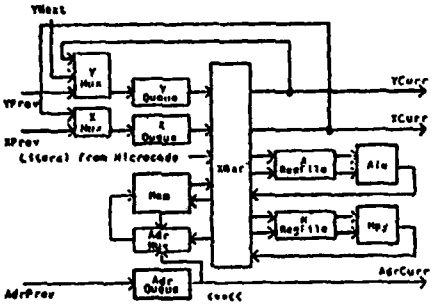


Figure 9: Warp cell datapath

3.6.2 Warp Road Following Algorithm

The Warp-implemented road following algorithm is very simple, but proved to be remarkably robust. The algorithm is region-based; it searches for the road as a bright region in the blue spectrum of a color image. A 100 x 512 band of the image is taken about halfway down the image. The algorithm then works as follows:

1. **Blue Filter.** The color image is filtered by digitizing only the blue band. Blue was chosen because blue is a strong component in the color of the roads we are driving on (asphalt and concrete), but less strongly a component of the background (generally grass).

2. **Edge-preserving smoothing.** This is a smoothing operation which avoids smoothing across edges. It is the algorithm EGPR in the Spider subroutine library [4], implemented on Warp. The algorithm takes a 5 x 5 window around each pixel and chooses nine subwindows in the 5 x 5 window. The subwindow with smallest variance is chosen, and the central pixel is replaced by the mean of this window. Two passes of this algorithm are executed. The effect is to remove noise from the image, especially noise from the poor quality of the TV reception in some cases.

3. **Histogramming.** A standard histogram is taken on the Warp machine.

4. **Threshold selection.** The histogram is used by the Sun 120 to select a threshold. The threshold is selected by starting at the 50th percentile level in the histogram and then finding a local minimum by comparing adjacent 3-element averages of the histogram.

5. **Binarization.** A gray value table translation table is constructed by the Sun using the threshold, and the image in binarized using this table on Warp.

6. **Region smoothing.** The resulting binary image is once again subjected to two passes of edge-preserving smoothing. The idea here is to remove small cracks in the road, and to eliminate small regions of ones in the background. Edge-preserving smoothing was chosen for this step instead of a more traditional operation, like shrinking and growing, because the edge-preserving filtering program was available while the (simpler) binary operator program was not.

7. **Blob detection.** At this point the road is a region of ones surrounded by a background of zeroes. Ten scan lines, taken ten rows apart, are taken from the image and each is examined to find the longest continuous sequence of ones. Each scan line thus defines a left and right road edge. The left and right edges are

averaged together individually to find the estimated road edges. An earlier approach was to find the left edge by finding the first long sequence of ones moving to the right from the left side of the image and the right edge similarly. This did not work as well as the second approach, since the vehicle tended to steer into the center of forks in the road.

8. **Steering.** Our servoing strategy is to steer the vehicle to keep the center of the road centered in the image. Basically we start with a large (512 x 512) image array and reduce it as quickly as possible to a point (x, y). This is the point considered to be the center of the road some fixed distance in front of the vehicle. It is also the point to which the vehicle steers. Assuming that the center of the image is the point $(0, 0)$, the steering command is to turn left or right at some $d\tau/dt = \gamma x$ where γ is a gain constant related to the distance ahead imaged and to vehicle speed. $d\tau/dt$ is rate of turn of the vehicle (giving path curvature) in degrees per second. See [6]for details.

3.6.3 Hardware Configuration

In addition to programming an efficient road following algorithm on Warp, we have made improvements in our video transmission system and vehicle interface that have increased the reliability of our system and further reducted time between image digitizations. Time reductions between in the image processing cycle increase the servo rate of the vehicle steering control loop, and enable the vehicle to drive at higher speed.

We chose to digitize the image of the blue band only, in order to obtain the highest possible contrast between the test road and the surrounding grass in the image. Since grass absorbs almost all blue light and the asphalt road reflects a lot of blue light, the TV image in the blue band shows a very bright road surrounded by very dark grass. The blue filtering of the signal is tied to the particular road on which we are testing the vehicle. The next step in hardware configuration improvement is to selectively digitize the red, green and blue bands and to combine them using our Matrox frame buffers and the Warp.

4. Conclusion

We have presented a comprehensive view of a vision-based road-following system for an autonomous vehicle. Various parts of this system exist and have been tested both off-line on "canned" images and during real-time tests using the Terregator.

An overall picture of our system can be seen by considering the path of a single image through the entire processing loop. First, the Map module announces a set of predictions for the current scene, knowing the vehicle's position. The Vision module then dynamically applies color and texture segmentation techniques to extract the predicted road region. An oriented edge tracker uses the geometry of the extracted road region and the predicted interest segments to either localize the position of the road or reject the region and report failure. If road or intersection region detection is successful, the Navigator is alerted and generates a steering plan from the road region. If not successful, the Vision system halts and signals the blackboard so that another module (or person) to take control. The steering plan is received by the low-level motion cor ol module, which interfaces to the vehicle's gyros and shaft encoders and executes the steering strategy. Timestamps on data carried through the entire system enable the vehicle to be controlled in real time, with old steering plans aborted as the Navigator creates new ones. To work for continuous motion road-following even at the slowest speed the Terregator has run in any road-following experiment (10 cm/sec) the entire processing loop must complete every 10 seconds.

Warp has proved to be a useful high-speed processor for vision tasks. An important advantage of Warp over other image processing computers is its floating-point capability. Many of the processes we have discussed, such as image rectification, color segmentation, and oriented edge tracking, are implemented as floating-point algorithms and can run efficiently on Warp. Using the Warp, we have already demonstrated one efficient and robust road-following algorithm.

5. Acknowledgements

We would like to thank the Civil Engineers Red Whittaker, Chuck Whittaker, Francois Blitz, Steve Berman and Kai Lee for developing and maintaining the Terregator vehicle. Thanks to Mike Blackwell and Kevin Dowling for hardware, system interface and video support. Chuck Thorpe managed much of the effort and Hans Moravec provided support and encouragement.

Currently, this project is funded in part by Carnegie-Mellon University, by the Office of Naval Research under contract number N00014-81-K-0503, by the Western Pennsylvania Advanced Technology Center, by Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539, and by Denning Mobile Robotics, Inc. Richard Wallace thanks NASA for supporting him with a NASA Graduate Student Researchers Program Fellowship Grant.

References

1. Arnould, E., Kung, H.T., Menzilcioglu, O. and Sarocky, K. A Systolic Array Computer. Proceedings of 1985 IEEE International Conference on Acoustics, Speech and Signal Processing, March, 1985, pp. 232-235.

2. Gross, T., Kung, H.T., Lam, M. and Webb, J. Warp as a Machine for Low-level Vision. Proceedings of 1985 IEEE International Conference on Robotics and Automation, March, 1985, pp. 790-800.

3. Kung, H.T. and Webb, J.A. Global Operations on the CMU Warp Machine. Proceedings of 1985 AIAA Computers in Aerospace V Conference, American Institute of Aeronautics and Astronautics, October, 1985.

4. Electrotechnical Laboratory. *SPIDER (Subroutine Package for Image Data Enhancement and Recognition).* Joint System Development Corp., Tokyo, Japan, 1983.

5. Stentz, A., Shafer, S., Thorpe, C. An Architecture for Sensor Fusion in an Autonomous Land Vehicle. forthcoming.

6. Wallace, R., Stentz, A., Thorpe, C., Moravec, H., Whittaker, W., Kanade, T. First Results in Robot Road Following. Proceedings of IJCAI 85, August, 1985.

7. W. Whittaker. Terrogator - Terrestrial Navigator. Carnegie-Mellon Robotics Institute, 1984.

8. Woo, B., Lin, L. and Ware, F. A High-Speed 32 Bit IEEE Floating-Point Chip Set for Digital Signal Processing. Proceedings of 1984 IEEE International Conference on Acoustics, Speech and Signal Processing, 1984, pp. 16.6.1-16.6.4.

Motion Control

Pulsewidth Modulation Control of Brushless DC Motors for Robotic Applications

PATRICK F. MUIR, STUDENT MEMBER, IEEE, AND CHARLES P. NEUMAN, SENIOR MEMBER, IEEE

Abstract—**Pulsewidth modulation (PWM) control of brushless dc motors is implemented with digital servo mechanisms for robotic applications. Under the assumption that the pulse period is much smaller than the motor time-constants, the motor is modeled by a discrete-time transfer function with the pulsewidth playing the role of the control signal. This model enables the application of classical linear control engineering to the design of a digital position servo for the brushless dc steering motors on the CMU Rover. The controller is implemented with a microprocessor and programmable timer to calculate concurrently the actuating signals, time sampling periods, and pulsewidths, as well as to provide commutation. Computer simulation and real-time hardware implementation of the servo demonstrate the efficacy of the approach.**

I. INTRODUCTION

THE DESIGN and implementation of digital servo controllers for brushless dc motors, utilizing pulsewidth modulation (PWM), has become a significant control engineering task because of the desirable characteristics of these motors for robotic applications. Brushless dc motors (using samarium-cobalt permanent magnets) are appropriate for robotic applications because of their high torque-to-weight ratio [1], ease of computer control, efficiency, and simple drive circuitry. Semiconductor power transistors can drive the motor directly from a microprocessor. Power transistors operate most efficiently in a switching mode. Velocity control of a brushless dc motor is accomplished (in the switching mode of operation) by the PWM of the stator coil voltages.

If the motor position is measured by a digital shaft encoder, the feedback control system, with the exception of the motor, is digital. The brushless dc steering motors on the CMU Rover [2] (described in Section III) exemplify such a system. The Rover is a mobile robot which rolls on three wheels that are actuated by brushless dc motors. In this paper, digital servo controllers are designed using PWM to provide mobility. The steering motors are modeled, position controllers are designed, and the control system is simulated and implemented in hardware. Simulation and experimental results demonstrate that the design goals of zero overshoot and a 100-ms settling time are achieved.

The PWM control of a linear analog system is assessed. Under the assumption that the pulse period is much smaller than the time-constants of the system, the system can be modeled by a linear discrete-time transfer function, with the pulsewidth playing the role of the control signal. This model enables the application of classical control engineering [3]–[6] to the design of pulsewidth-modulated systems for the control

Manuscript received April 25, 1984. This paper was supported by an R. K. Mellon Fellowship granted to P. F. Muir by Carnegie-Mellon University, the Office of Naval Research under Contract N00014-81-0503, and the Department of Electrical and Computer Engineering, Carnegie-Mellon University.

The authors are with the Department of Electrical and Computer Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

0278-0046/85/0800-0222$01.00

of brushless and conventional brushed dc motors, and electromagnetic solenoids.

This paper is organized as follows. The operation of brushless dc motors is reviewed in Section II, and their application on the CMU Rover is described in Section III. PWM control of linear analog systems is highlighted (in Section IV) and applied to model the steering motor on the Rover using experimental data (in Section V). This modeling process and the ensuing controller design are accomplished entirely in the discrete-time domain. An algorithm is then presented for transforming the discrete motor model into an equivalent model at a sampling period which is different than the sampling period of the experimental data, since the sampling period of the controller is not specified at the identification stage. Consequently, when a low-order transfer function is identified from the original data, the modeling experiments need not be repeated at the controller sampling period to reidentify the model.

Controller design (including the choice of sampling period) is outlined in Section VI. The controller sampling period is specified in terms of processing time, motor response time, velocity resolution, and timer operational limitations. Because the servo execution time exceeds one-half of the sampling period, the processing time is incorporated (as a computational delay) in the closed-loop system model, thereby increasing the order of the system. Nonlinearities in the control system (caused by friction, motor saturation, and position quantization) are neglected in the controller design. Controller gains are calculated to satisfy the design goals of zero overshoot and a 100-ms settling time. The step-response of the closed-loop control system, using these gains, is simulated in the presence of the aforementioned nonlinearities. The controller gains which meet the performance specifications (in the presence of the nonlinearities) are selected for the hardware evaluation.

The hardware implementation of the controller is evaluated in Section VII. Motorola 6805 microprocessors execute the control algorithms, which are stored in nonvolatile read-only-memory. An interrupt driven routine and a programmable timer enable the processor to calculate concurrently the actuating signal and time sampling periods, and to provide pulsewidth modulation. The performance of the position servo is evaluated from experimental step-response data. The results are summarized and concluding remarks are advanced in Section VIII.

II. BRUSHLESS DC MOTORS

A brushless dc motor has the same torque-speed characteristic as a conventional dc motor even though the principle of operation is more complex [7]. There is no electrical connection to the rotor of a brushless dc motor because the rotor consists of permanent magnets. Samarium–cobalt permanent magnets, which provide higher torque than conventional alnico magnets, are commonly used in brushless dc motors. Commutation of a brushless dc motor is accomplished by electronically switching the current in the stator windings. The proper stator winding polarities (at each instant) are derived from the shaft position, as read from a shaft encoder, and the desired direction of rotation. Velocity control is accomplished either by adjusting the stator currents (using D/A converters and current amplifiers), or, more simply, by adjusting the current duty cycle (using power transistors and PWM). To reverse the direction of rotation, the stator windings are sequenced in reverse order, rather than reversing the current polarity.

Even though the operation of a brushless dc motor is more complex than that of a conventional brushed dc motor, practical advantages accrue. The removal of heat produced in the windings of brushless dc motors is more easily accomplished because the path to the environment is shorter. Problems with brushes, such as wear and brush noise, are eliminated. Brushless dc motors require minimal interface circuitry for microprocessor control. Power transistors are operated in a switching mode, as coil drivers are more efficient than the analog power amplifiers used with conventional motors. Minimizing weight and power consumption is essential for mobile robots because the capacity of self-contained power sources is limited. Disadvantages of brushless dc motors are the need for electronic commutation, its high cost, and low availability. As the demand for brushless dc motors grow, these motors will become more available and less expensive.

III. THE CMU ROVER

The CMU Rover [2] is a mobile robot currently being designed and constructed in the Robotics Institute of Carnegie-Mellon University, Pittsburgh, PA. The CMU Rover is cylindrical in shape, 1-m tall, and 55 cm in diameter. Mobility is provided by three wheels upon which the robot is supported. Three brushless dc steering motors [8] control the orientation of the wheels and three additional brushless dc drive motors control the rotation of the wheels. The motors are directly coupled to the wheels. A Motorola 6805 microprocessor [9] is dedicated to the control of each motor. Servo reference positions are communicated to the individual motor processors via a common serial line from high-level processes [2] executing on independent onboard processors. Power MOSFET devices drive the motor coils from the microprocessor output ports through optoisolators which protect the processor from electrical noise generated in the motor. The motor shaft position is fed back to the processor via a digital shaft encoder [10].

IV. PWM OF A LINEAR SYSTEM

There are practical reasons why the dynamic models of dc motors cannot be applied directly to model the motors on the CMU Rover. Although many of the characteristic parameters are provided by the motor manufacturer, there are parameters (e.g., the moment of inertia of the load, frictional torque, and damping constant) that must be obtained experimentally after the motor is built into the robot. Furthermore, the input to a conventional dc motor is the voltage applied to the motor windings; whereas, the voltage pulsewidth plays the role of the input for a motor controlled using PWM.

The PWM control of motors is analyzed for the state-space model of the Nth-order linear time-invariant system

$$\frac{dx}{dt} = Ax(t) + bu(t) \tag{1}$$

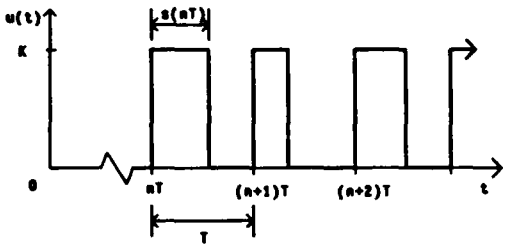


Fig. 1. Pulsewidth modulation.

where the $(N \times 1)$ state vector is x and the scalar input is u. The $(N \times N)$ motor matrix is A and the $(N \times 1)$ input vector is b. The solution of (1) is [5]

$$x(t) = \exp\{A(t-t_0)\}x(t_0) + \int_{t_0}^{t} \exp\{A(t-\lambda)\}bu(\lambda)\ d\lambda \tag{2}$$

where $\exp\{At\}$ is the matrix exponential [3, 5].

The scalar pulsewidth modulated signal $u(t)$ is shown in Fig. 1. The input $u(t)$ is the constant K (volts) for the fraction s/T of each period, and zero for the remainder of each period. The pulsewidth is the *magnitude* of the control signal and is, therefore, positive. Negative control signals reverse the commutation sequence of the motor (as discussed in Section VI). The goal is to find conditions under which (2) is *linear* in the pulsewidth s. The digital controller samples the states at discrete-time instants. Instead of the continuous state vector $x(t)$, attention focuses on the state vector $x(nT)$ at the sampling instant nT, where T is the constant sampling period and n is the iteration index. In (2), the sampling period, from $t = nT$ to $t = (n+1)T$, is divided into two subperiods. The first runs from $t_0 = nT$ to $t = nT + s(nT)$; where the pulsewidth $s(nT)$ can vary from sampling period to sampling period, and the pulse height is constant. In the second, from $t_0 = nT + s(nT)$ to $t = (n+1)T$, the input $u(t)$ is zero. Thus

$$x[nT + s(nT)] = \exp\{As(nT)\}x(nT) + K\int_0^{s(nT)} \exp\{A\lambda\}\ d\lambda\ b \tag{3}$$

and

$$x[(n+1)T] = \exp\{A[T - s(nT)]\}x[nT + s(nT)]. \tag{4}$$

Upon substituting (3) into (4), the state-vector $x[(n+1)T]$, at the $(n+1)$th sampling instant, is related to the state-vector $x(nT)$ according to

$$x[(n+1)T] = \exp\{AT\}x(nT) + K\ \exp\{AT\}\ \exp\{-As(nT)\} \cdot \int_0^{s(nT)} \exp\{A\lambda\}\ d\lambda\ b. \tag{5}$$

To continue the development, the matrix exponentials in (5) are approximated by their first-order series expansions [4]; i.e., $\exp\{At\} \simeq I + At$, under the assumption that the sampling period T and consequently the pulsewidth $s(nT) \leq T$ are much *smaller* than the system time-constants. In first-order systems, this assumption ensures that the scalar exponential $\exp\{T/\tau\}$ can be adequately approximated by $1 + T/\tau$, where τ is the system time-constant. By applying a similarity transformation [3] to diagonalize the system matrix A, the first-order condition generalizes for approximating the matrix exponentials in (5). Upon substituting the first-order matrix approximation and retaining the linear terms in $s(nT)$, (5) leads to

$$x[(n+1)T] = \{I + AT\}x(nT) + Kbs(nT) \tag{6}$$

where I is the $(N \times N)$ identity matrix. The discrete state-space PWM model in (6) is *linear* in the pulsewidth $s(nT)$ which plays the role of the control signal. The state, and hence the outputs (which are linear combinations of the states), depend linearly on the pulsewidth $s(nT)$. The only assumption made in leading to (6) is that the sampling period is much smaller than the time-constants of the system. This assumption is practical because conventional digital control systems operate on a sampling period which is much smaller than the response time of the system under control. This engineering assumption and interpretation of the linear model in (6) lay the foundation for the design (in Section VI) of control systems for the motors on the CMU Rover.

V. MODELING THE STEERING MOTOR

A. Introduction

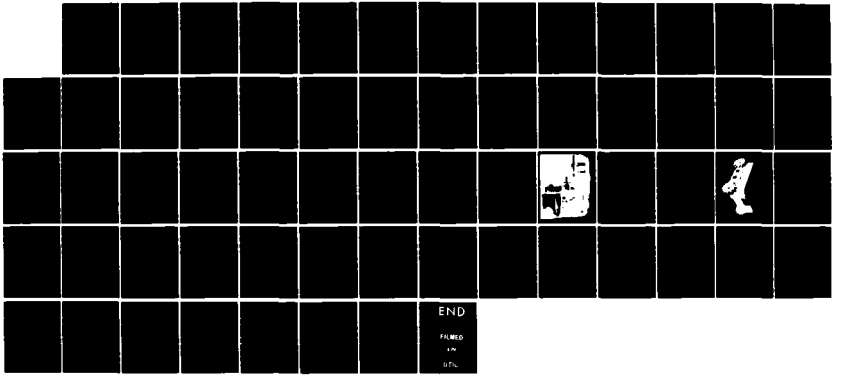
The framework of Section IV is applied to the practical problem of modeling the brushless dc steering motors on the CMU Rover. The analog transfer function, from input voltage to output velocity of a dc motor is linear [7]. Consequently, the motor under PWM control can be characterized by the linear discrete-time state-space model in (6), and a corresponding linear transfer function, from pulsewidth to velocity, if the sampling period is small compared to the time-constants of the motor. Since the motor parameters are unknown, experimental data are acquired (in this section) to identify the discrete-time model. The order of the model is chosen to ensure acceptable accuracy, without increasing the complexity of the servo controller.

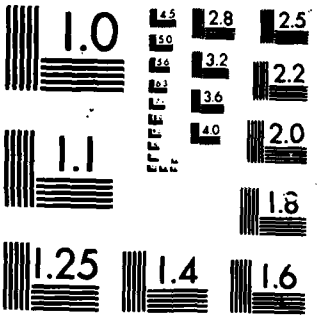
B. Experimental Data

The velocity step-response of a steering motor is easily measured and sufficient to identify the transfer function (from pulsewidth to velocity). Velocity measurements are acquired every 2 ms, since this is a convenient sampling period to implement. Data are taken until the step-response settles (160 data points are stored for model identification). The dominant time-constant of the motor is found to be 58 ms. The motor exhibits nonlinear saturation at the maximum velocity (6.25 revolutions per second) and a frictional dead zone at small command inputs. The data used to identify the model are taken at a command value that is within the active linear range of motor operation. The transfer function selected to model the motor has the simplest structure which closely approximates the experimentally obtained step-response of the motor.

AD-A164 404 AUTONOMOUS MOBILE ROBOTS(U) CARNEGIE-MELLON UNIV 2/2
PITTSBURGH PA MOBILE ROBOT LAB H P MORAVEC 30 JAN 86
CMU-RI-MRL-86-1 N00014-81-K-0503

UNCLASSIFIED F/G 13/6 NL





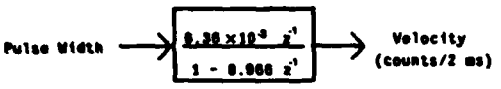


Fig. 2. Steering motor model.

C. Model Order

The input–output transfer function of a conventional or brushless dc motor, from voltage input to velocity output is second-order [7]. The discrete model (6) of the motor under PWM control is also second-order. One mode of the motor dynamic response is characterized by its mechanical time-constant and the second mode by its electrical time-constant. Since the electrical time-constant of the motor is much smaller than the mechanical time-constant, a first-order model should be sufficiently accurate for controller design.

First- and second-order discrete-time transfer functions are introduced to model the steering motor (from pulsewidth input to velocity output). The first-order transfer function is

$$G_1(z^{-1}) = \frac{K_1 z^{-1}}{1 - p z^{-1}} \tag{7}$$

and the second-order model is

$$G_2(z^{-1}) = \frac{K_2 z^{-1}(1 + z_0 z^{-1})}{1 - p_1 z^{-1} - p_2 z^{-2}}. \tag{8}$$

A computer program was written to simulate the step-response of these models using user-specified model parameters (i.e., K_1, and p; and K_2, z_0, p_1 and p_2). The program calculates the accumulated squared-error between the simulated output of each model and the experimentally obtained step-response. The user systematically adjusts the model parameters to reduce the accumulated squared-error for both the first- and second-order models. Finally, the minimum squared-error of the first-order model is compared with the minimum squared-error of the second-order model to decide whether the second-order model is significantly more accurate to warrant the additional implementational complexity.

The second-order model of the steering motor produces a squared-error which is only 4.7 percent less than that of the first-order model. This small improvement, in our opinion, does not warrant its corresponding increased complexity.

D. Identified Steering Motor Model

The transfer function model $G_1(z^{-1})$ of the steering motor, which is used in the controller design (in Section VI), is depicted in Fig. 2. The motor velocity is measured in units of shaft encoder counts (there are $2^{12} = 4096$ counts/revolution) per sampling period (2 ms). The model has a dc gain of 0.187 and a pole at $z = 0.966$ corresponding to a time-constant of 58 ms. The second-order model has the same dc gain, poles at $z = 0.965$ and $z = 0.436$ (corresponding to time-constants of 56 ms and 2 ms, respectively), and a zero at $z = 0.397$. Since the pole at $z = 0.436$ responds much faster than the dominant pole at $z = 0.965$, which matches the pole of the first-order model, the response of the first-order model closely resembles that of the second-order model.

E. Sampling Period of the Model

The sampling period of the motor controllers is not specified when experimental data are collected to model the motors. The controller sampling period may differ from the sampling period of the experiments. Since a discrete transfer function model of an analog system is an explicit function of the sampling period [11], the discrete motor model used in the controller design must correspond to the controller sampling period.

To change the sampling period of the motor model, the discrete transfer function $G_1(z^{-1})$ in (7) is assumed to be the step-invariant transformation [11] of the first-order analog transfer function

$$G(s) = \frac{K}{\tau s + 1}. \tag{9}$$

Thus

$$p = p(T) = \exp\{-T/\tau\} \tag{10}$$

and

$$K_1 = K_1(T) = K[1 - p(T)]. \tag{11}$$

When the sampling period is changed from T to T_1, the digital transfer function in (9) becomes

$$G_1(z^{-1}) = \frac{K_1(T_1) z^{-1}}{1 - p(T_1) z^{-1}} \tag{12}$$

where

$$p(T_1) = \exp\{T_1/T \ln p(T)\} \tag{13}$$

and

$$K_1(T_1) = K_1(T) \frac{1 - p(T_1)}{1 - p(T)}. \tag{14}$$

VI. CONTROL SYSTEM DESIGN

A. Introduction

The objective of this section is to design a position servo for the steering motor. The linear discrete-time transfer function model identified in Section V enables the application of classical linear control engineering to PWM controller design. The design goals are zero overshoot and a 100-ms settling time.

B. Sampling Period

Motor characteristics and processor capabilities lead to the selection of the controller sampling period. The controller must operate with a sampling period that is much smaller (e.g., 10 times smaller) than the motor time-constants, so that the pulsewidth-modulated motor can be modeled by the discrete transfer function in Fig. 2. Since the time-constant of the steering motor is 58 ms, the controller sampling period should not exceed 5.8 ms. Execution of a prototype servo program is timed and found to set a lower limit on the sampling period at 1.27 ms, because the program must be able to execute within each sampling period. The minimum

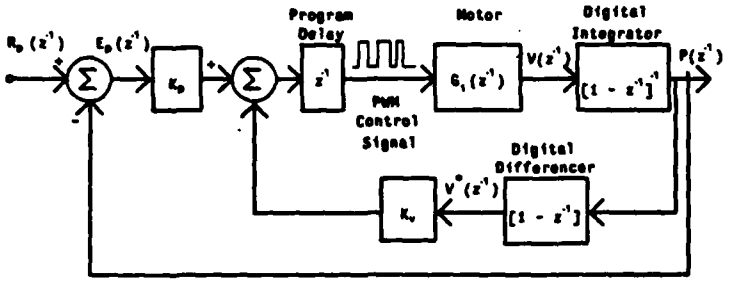


Fig. 3. Steering motor position servo controller.

sampling period is also limited by the precision of velocity calculations. Velocity measurement precision is low if the sampling period is small, because velocities are calculated as the difference between position readings at successive sampling periods. By experimentation with the prototype servo program, the lower limit (1.85 ms) on the sampling period is found to provide sufficient velocity precision and thereby avoid undesirable nonlinear quantization effects which result in jerky motor operation. The controller sampling period of 2 ms is chosen because it satisfies the aforementioned constraints and because it is convenient to implement sampling periods that are multiples of 0.25 ms with the programmable timer. Since the pulse period of the PWM is one sampling period, the choice of 2 ms as the sampling period guarantees that the linear modeling assumption of Section IV (i.e., the pulsewidth is much smaller than the time constant 58 ms of the motor) is satisfied.

C. Control System Structure

The position servo (in Fig. 3) is implemented by incorporating position and velocity feedback. The control signal is the pulsewidth modulated voltage applied to the motor coils. The pulsewidth in the nth sampling period is the magnitude of $s(n)$, where

$$s(n) = K_p\{R_p(n-1) - P(n-1)\} - K_v V^*(n-1) \tag{15}$$

and where

$R_p(n-1)$	current reference motor position,
$P(n-1)$	current shaft position as read from the shaft encoder,
$E_p(n-1)$	current position error,
$V^*(n-1)$	current velocity calculated as $[P(n-1) - P(n-2)]$,
K_p	position gain (in Section VI-D),
K_v	velocity gain (in Section VI-D).

The position and velocity gains K_p and K_v control the transient response of the servo. The height of each pulse is constant (24 V) and the pulsewidth is calculated as the *magnitude* of (15). The *sign* of (15) specifies (in Section VII) the motor coil commutation sequence. This is analogous to reversing the polarity of the voltage applied to a brushed dc motor. The delay z^{-1} is introduced in the forward path to model the execution time of the controller program. The calculation of the control signal is not completed until 1.27 ms after the inputs are received, due to the program execution time (as explained in Section VI-B). To ensure that the actuating signal is synchronized with the sampling period, the calculated control signal is stored until the beginning of the next sampling period, when the magnitude of the control signal is used as the pulsewidth and the sign specifies the commutation sequence. The motor parameters K_1 and p are calculated at the controller sampling period of 2 ms using the formulae in Section V-E. In this design, the controller sampling period and sampling period of the modeling experiments coincide and the transfer function in Fig. 2 is applied for the controller design.

D. Gain Calculation

The closed-loop transfer function of the position servo (in Fig. 3) is third-order

$$\frac{P(z^{-1})}{R_p(z^{-1})} = \frac{K_p K_1 z^{-2}}{1-(p+1)z^{-1}+[p+K_1(K_v+K_p)]z^{-2}-K_1 K z^{-3}}. \tag{16}$$

The controller gains K_p and K_v are calculated to meet the design specifications of zero overshoot and a 100-ms settling time. The transfer function in (16) is factored into the *cascade* of a second-order component and a first-order component

$$\frac{P(z^{-1})}{R_p(z^{-1})} = \frac{K_3 z^{-1}}{(1-\alpha z^{-1})^2}\,\frac{z^{-1}}{(1-p_3 z^{-1})}, \quad \text{where } 0<p_3<\alpha<1. \tag{17}$$

The objective is to force the critically damped second-order component (with two equal real poles at $z = \alpha$) to dominate the closed-loop response. The closed-loop system is thus designed to respond as fast as possible without overshoot. By equating (16) and (17), the third system pole p_3, feedback gains K_v and K_p, and gain K_3 are computed in terms of α and the motor constants K_1 and p according to

$$p_3 = (p+1) - 2\alpha \tag{18}$$

$$K_v = \frac{p_3\alpha^2}{K_1} \tag{19}$$

$$K_p = \frac{\alpha^2 + 2\alpha p_3 - p - K_1 K_v}{K_1} \tag{20}$$

and

$$K_3 = K_p K_1. \tag{21}$$

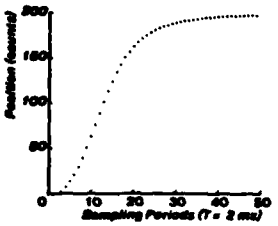


Fig. 4. Simulated step-response of steering motor position servo.

The settling time of the closed-loop system is then calculated from (17) for different values of α. The servo gains K_p and K_v are calculated from (19) and (20) for values of α which produce settling times less than 100 ms. The choice of gains is finalized by simulating the control system, with the calculated gain combinations, on a computer in the presence of nonlinear motor saturation and quantized position feedback values. The feedback gain values

$$K_v = 32 \text{ and } K_p = 3 \tag{22}$$

provide acceptable simulated response characteristics and satisfy the design constraints in computer simulation. The value of $\alpha = 0.838$ (corresponding to a time-constant of 11.3 ms) is substituted into (18) to calculate the location of the third pole $p_3 = 0.290$ (corresponding to a time-constant of 1.6 ms). The third pole thus responds much faster than the two equal dominant poles, as desired.

E. Control System Simulation Results

The simulation program implements the block diagram of Fig. 3 to calculate (at discrete time instants) the step-response of the steering motor servo. The simulated step-response of the steering motor position servo controller, using the gains in (22), is shown in Fig. 4. The step-response does not overshoot and displays a 100-ms settling time, and thus satisfies the design specifications (with zero steady-state error).

VII. HARDWARE IMPLEMENTATION AND EXPERIMENTAL RESULTS

A. Hardware Overview

The steering motor controller is implemented as an assembly language program running in real time on a Motorola 6805 microprocessor. Reference positions are communicated (over a serial communication link) to the processor from high-level processors. The processor communicates, the pulsewidth-modulated control signal to the motor via an output port to the motor coil drivers. The motor shaft position is fed back to the input port of the processor from an optical shaft encoder. In each sampling period, the program calculates the pulsewidth and the motor coil excitation pattern, and produces a pulsewidth-modulated signal to control the motor.

B. Controller Program

Two independent programs are shown in the flowchart of the servo program in Fig. 5. The main routine implements the calculations and logic which produce the motor coil excitation pattern (i.e., commutation) and actuating signal (i.e., pulsewidth), and requires approximately 1.27 ms of each 2-ms sampling period to execute. The interrupt routine handles only those functions that require accurate timing, such as reading the shaft encoder, sending signals to the motors, and controlling the timer. The software is structured so that the most urgent tasks (those serviced in the interrupt routine) are processed when necessary, and the tasks for which timing is not critical (those serviced in the main routine) use the remaining processing time.

The programmable timer is used to time the pulsewidth and sampling period, and synchronize the control signals in the following manner. The timer is loaded with the pulsewidth (which was calculated by the main routine in the previous sampling period); and the proper motor coils are energized by loading the microprocessor output port with the excitation pattern (the excitation pattern was also determined by the main routine in the previous sampling period); and the position of the motor shaft is stored. The timer counts down the pulsewidth, while the main routine calculates the pulsewidth and coil excitation pattern for the next sampling period. When the pulsewidth has elapsed, the timer generates a hardware interrupt to the processor. The processor immediately stores the present state of execution of the main routine and begins executing the interrupt routine. The interrupt routine calculates the time remaining in the sampling period, loads this value in the timer, and turns off all of the motor coils by storing a 0 in the output port. Control is then returned to the main routine, which resumes execution at the point at which it was interrupted. After the programmable timer has counted down the remaining time in the sampling period, a second interrupt is generated. By this time, the main routine has completed its calculations, and the cycle repeats each succeeding sampling period.

Implementation of the multiplication operation in assembly language code is accomplished using shift and add instructions. Addition and subtraction of 12-bit quantities on the 8-bit processor is achieved by double-precision calculations. Calculations involving cyclical shaft position readings must be checked and corrected for wraparound errors. Position readings must lie within the range 0–4095. If the calculated position error is outside this range, a multiple of 4096 must be added to or subtracted from the value (as appropriate), to bring the result within the allowable range. A similar correction procedure must be executed if the calculated velocity value is outside of the range -2048 to 2048.

The main program implements electronic commutation of the motor coil voltages by a table look-up to determine the excitation pattern which produces the maximum torque in the desired direction for the present shaft position. The table is a list of ranges of shaft positions; each with two associated motor-coil excitation patterns. The first excitation pattern produces maximum motor torque in the clockwise direction if the motor position is within the range. The second produces maximum torque in the counterclockwise direction. The range in which a shaft position occurs is identified by comparing the shaft position with the range boundary positions. If the shaft position is greater than or equal to the lower boundary of a range *and* less than the upper boundary, then

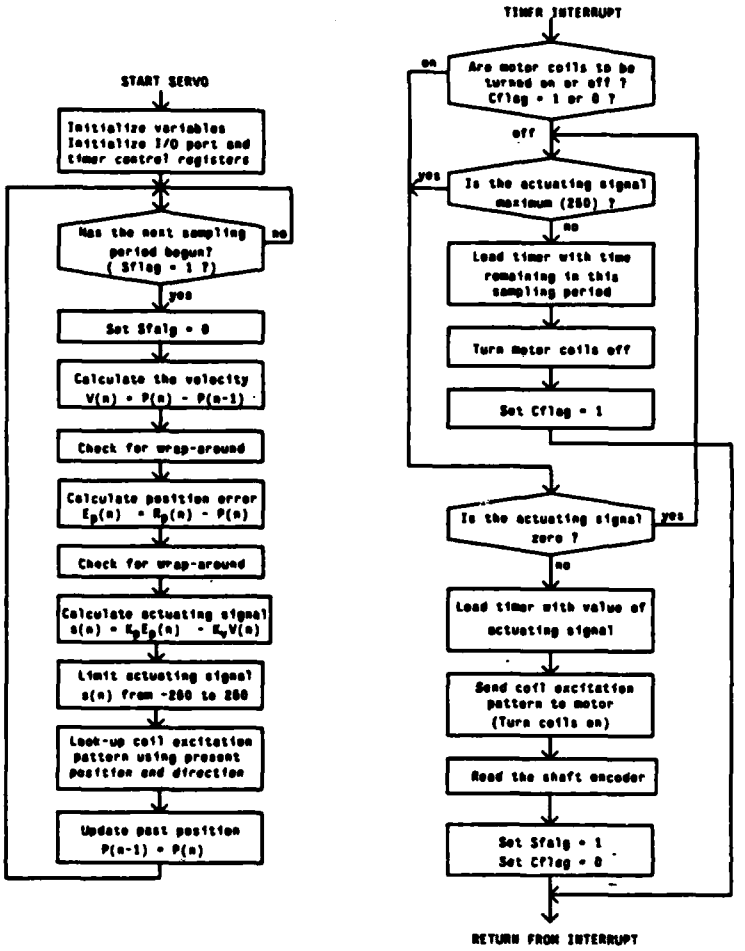


Fig. 5. Flowchart of assembly language servo program.

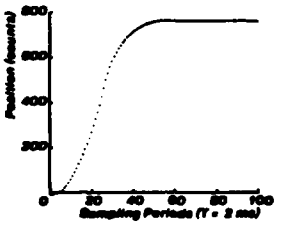


Fig. 6. Hardware step-response of steering motor servo controller.

the shaft position is in that range. The excitation value for the desired direction of rotation is read from the table under the entry for that range.

C. Experimental Results

A typical experimentally obtained step-response of the steering motor servo controller (with the Rover stationary and only one motor operating) is plotted in Fig. 6. The plot shows that the servo response satisfies the design specifications of zero overshoot and a 100-ms settling time. The shape of the response is similar to the simulated step-response plot in Fig. 4. Neither plot exhibits third-order characteristics, which verifies that, by design, the third system pole p_3 in (17) responds significantly faster than the two dominant equal poles α.

VIII. CONCLUSION

The modeling, design, and implementation of a controller utilizing a pulsewidth-modulated actuating signal is highlighted in this paper. A brushless dc motor (actuated by a pulsewidth-modulated signal) is modeled (using experimental data) as a discrete linear system whose control signal is the pulsewidth, under the assumption that the pulse period is much smaller than the time-constants of the motor. The controller sampling period and PWM pulse period are equal in this implementation. This model enables the application of classical linear control engineering to the design of a digital controller for the motor.

A position servo controller designed for the steering motors on the CMU Rover meets the specified performance objectives. The controller is implemented on a microprocessor which uses a programmable timer and an interrupt driven routine, and calculates the pulsewidth, provides commutation, and times concurrently the sampling period and pulsewidth.

Simulated and experimental step-response data demonstrate that the desired servo operation is realized.

The servo can be enhanced by measuring the shaft encoder pulse period to provide a more precise velocity measurement [12]. The position servo on the CMU Rover steering motors has recently been modified to servo simultaneously to a desired position and velocity [13]. The framework of this paper can be applied to the PWM control of brushless dc motors for robotic manipulators, conventional brushed dc motors, and electromagnetic solenoids.

ACKNOWLEDGMENT

The authors express their appreciation to Dr. Hans P. Moravec, Head of the Rover Project (Robotics Institute, Carnegie-Mellon University), for his support and cooperation throughout the course of their research.

REFERENCES

[1] H. Asada and T. Kanade, "Design of direct-drive mechanical arms," The Robotics Institute, Carnegie-Mellon University, Tech. Rep. CMU-RI-TR-81-1, Pittsburgh, PA, Apr. 1981.

[2] H. P. Moravec, "The Stanford Cart and The CMU Rover," *Proc. IEEE*, vol. 71, pp. 872–884, July 1983.

[3] T. Kailath, *Linear Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[4] G. F. Franklin and J. D. Powell, *Digital Control of Dynamic Systems*. Reading, MA: Addison-Wesley, 1981.

[5] J. J. D'Azzo and C. H. Houpis, *Linear Control System Analysis and Design*. New York: McGraw-Hill, 1981.

[6] B. C. Kuo, *Digital Control Systems*. Champaign, IL: SRL Pub. Co., 1977.

[7] Electro-Craft Corp., *DC Motors Speed Controls Servo Systems*. Hopkins, MN: Electro-Craft Corp., 1980.

[8] *BM-3201 Catalog Data*, Inland Motor Div., Kollmorgen Corp., Radford, VA, Sept. 1977.

[9] *CMOS 8-Bit Microprocessor, Advance Information*, Motorola Semiconductor Products Inc., Austin, TX, 1980.

[10] *Model R-2000 High Performance Modular Encoder*, Bull. R-2000, Renco Corp., Goleta, CA, Dec. 1981.

[11] C. P. Neuman and C. S. Baradello, "Digital transfer functions for microprocessor control," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-9, pp. 856–860, Dec. 1979.

[12] K. V. Rangan, "Position and velocity measurement by optical shaft encoders," The Robotics Institute, Carnegie-Mellon University, Tech. Rep. CMU-RI-TR-82-8, Pittsburgh, PA, June 1982.

[13] P. F. Muir, "Digital servo controller design for brushless dc motors," Master's proj. rep., Dep. of Elect. and Computer Eng., Carnegie-Mellon University, Pittsburgh, PA, Apr. 1984.

Kinematic Modeling Of Wheeled Mobile Robots (A Summary)

Patrick F. Muir [†] **and Charles P. Neuman** [‡]
Department of Electrical and Computer Engineering
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

We summarize our methodology for formulating the *kinematic equations-of-motion* of a wheeled mobile robot. The complete paper[1], which is currently being prepared for publication, is over one-hundred pages in length. Wheeled mobile robots having *conventional*, *omnidirectional*, and *ball wheels* are modeled. While our approach parallels the kinematic modeling of stationary manipulators, we extend the methodology to accommodate such special characteristics of wheeled mobile robots as *multiple closed-link chains*, *higher-pair contact points* between a wheel and a surface, and *unactuated* and *unsensed degrees-of-freedom*. We apply the *Sheth-Uicker* convention to assign coordinate axes and develop a *matrix coordinate transformation algebra* to derive the equations-of-motion. We calculate the *forward* and *inverse* solutions and interpret the conditions which guarantee their existence. Applications of the kinematic model are also described.

[†] Graduate student, Department of Electrical and Computer Engineering; and Member, Autonomous Mobile Robots Laboratory, The Robotics Institute.

[‡] Professor of Electrical and Computer Engineering.

1. Introduction

The wheeled mobile robot literature shows that the documented investigations have concentrated on the application of mobile platforms to perform intelligent tasks rather than on the development of methodologies for analyzing, designing, and controlling the mobility subsystem. Improved mechanical designs and mobility control systems will enable the application of WMRs to tasks were there are no marked paths and for autonomous mobile robot operation. A kinematic methodology is the first step towards achieving these goals.

Even though the methodologies for modeling and controlling stationary manipulators are applicable to WMRs, there are inherent differences which cannot be addressed with these methodologies, such as:

1.) WMRs contain multiple *closed-link chains*; whereas, manipulators form closed-link chains only when in contact with stationary objects.

2.) The contact between a wheel and a planar surface is a *higher-pair*; whereas, manipulators contain only lower-pair joints.

3.) Some degrees-of-freedom of a wheel on a WMR are not actuated or sensed; whereas, all degrees-of-freedom of each joint of a manipulator are actuated and sensed.

Wheeled mobile robot control requires a methodology for modeling, analysis and design which extends the principles applied to stationary manipulators. In this paper, we advance the kinematic modeling of WMRs, from the motivation of the kinematic methodology, to its development and applications. In Section 2, we present the three wheels (conventional, omnidirectional and ball wheels) utilized in all existing and foreseeable WMRs. We present a definition of a wheel mobile robot and enumerate our assumptions in Section 3. Coordinate systems are assigned to prescribed positions on the the robot (Section 4). We develop transformation matrices to characterize the translations and rotations between coordinate systems (Section 5). Matrix coordinate transformation algebra is developed as a means of calculating position, velocity, and acceleration relationships between coordinate systems in Section 6. We apply the axioms and corollaries of this algebra to model the kinematics of WMRs.

The equations-of-motion relating robot positions are developed in Section 7, and we develop the velocity and acceleration relationships in Section 8. We relate the motion of a wheel to the motion of the robot body by calculating a wheel Jacobian matrix. From the simultaneous motions of the wheels, we obtain the motion of the robot in Section 9. Specifically, we obtain the inverse solution, and the forward solution. We discuss the application of the kinematic methodology in

Section 10 and summarize the kinematic modeling procedure in Section 11. We outline our plans for continued research in Section 12.

Many sections and details of the original paper had to be omitted from this summary for brevity. The full paper contains: a survey of documented WMRs, detailed derivations of the inverse and forward solutions, detailed applications, the development of the kinematic model of several example WMRs, and a nomenclature and symbolic representation for WMRs. Further details on the topics presented in this summary are also included.

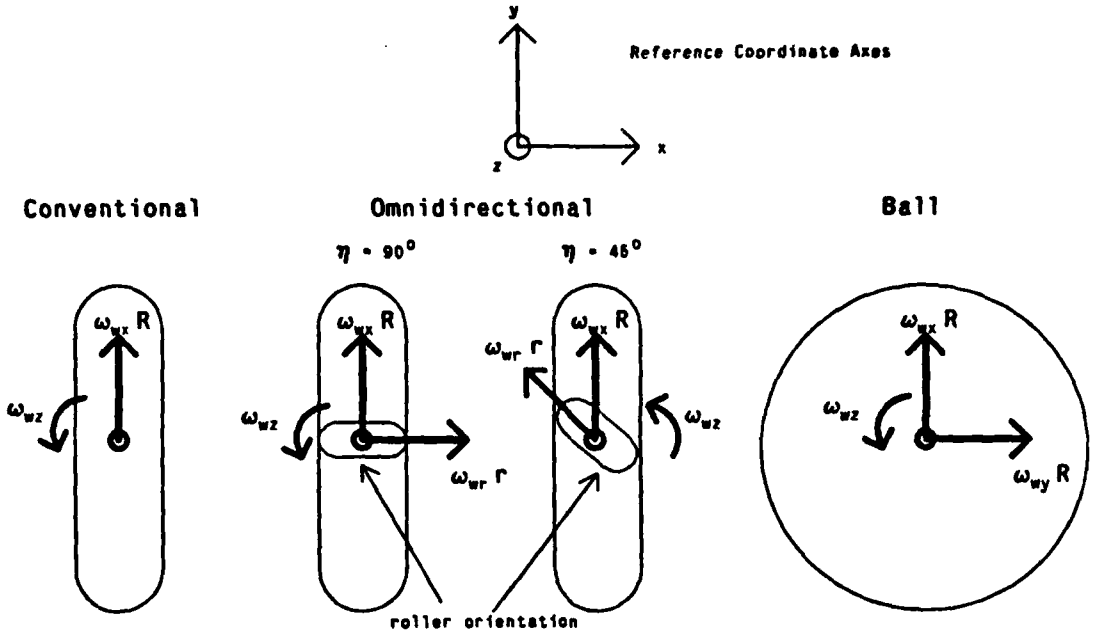
2. Wheel Types

Three basic types of wheels are used in WMRs: conventional, omnidirectional, and ball wheels. In addition, conventional wheels often are mounted on a steering link to provide an additional degree-of-freedom. The degrees-of-freedom of each wheel are indicated by the arrows in Figure 1. The kinematic equations relating the angular velocity of the wheel to its linear velocity along the surface of travel are also compiled in the figure.

The nonsteered *conventional* wheel is the simplest to construct having two degrees-of-freedom. It allows travel along a surface in the direction of the wheel orientation, and rotation about the point-of-contact between the wheel and the floor. We note that the rotational degree-of-freedom is slippage, since the point-of-contact is not stationary with respect to the floor surface. Even though we define the rotational slip as a degree-of-freedom, we do not consider slip transverse to the wheel orientation a degree-of-freedom, because the magnitude of force required for the transverse motion is much larger than that for rotational slip.

The *omnidirectional* wheel has three degrees-of-freedom. One degree-of-freedom is in the direction of the wheel orientation. The second degree-of-freedom is provided by motion of rollers mounted around the periphery of the main wheel. In principle, the roller axles can be mounted at any nonzero angle η with respect to the wheel orientation. The third degree-of-freedom is rotational slip about the point-of-contact. It is possible, but not common, to actuate the rollers of an omnidirectional wheel, with a complex driving arrangement.

The most maneuverable wheel is a *ball* which is actuated to posses three degrees-of-freedom without slip. Schemes have been devised for actuating and sensing of ball wheels, but we are unaware of any existing implementations. An omnidirectional wheel which is steered about its point-of-contact is kinematically equivalent to a ball wheel, and may be a practical design alternative.



Wheel Degrees-of-Freedom

Conventional	Omnidirectional	Ball
$V_y = \omega_{wx} R$	$V_y = \omega_{wx} R - \omega_{wr} r \cos(\eta)$	$V_y = \omega_{wx} R$
$V_x = 0$	$V_x = \omega_{wr} r \sin(\eta)$	$V_x = \omega_{wy} R$
$\omega_z = \omega_{wz}$	$\omega_z = \omega_{wz}$	$\omega_z = \omega_{wz}$

Legend

V_x and V_y	=	x and y components of the linear velocity of the wheel at the point-of-contact
ω_z	=	z component of the angular velocity of the wheel at the point-of-contact
ω_{wr}	=	angular velocities of the roller about their axles
ω_{wx} , ω_{wy} , ω_{wz}	=	x, y, and z angular velocities of the wheel about its center
η	=	angle of the roller axles with respect to the wheel orientation
R and r	=	radii of a wheel and a roller

Figure 1

Wheel Equations of Motion

3. Definitions And Assumptions

We introduce an operational definition of a WMR to specify the range of robots to which the kinematic methodology presented in this paper applies.

Wheeled Mobile Robot - A robot capable of locomotion on a surface solely through the actuation of wheel assemblies mounted on the robot and in rolling contact with the surface. A wheel assembly is a device which provides or allows relative motion between its mount and a surface on which it is intended to have a single point of *rolling* contact.

Each wheel (conventional, omnidirectional or ball wheel) and all links between the robot body and the wheel constitute a wheel assembly. We introduce the following practical assumptions to make the modeling problem tractable.

Assumptions

1.) The WMR does not contain flexible parts.
2.) The WMR moves on a planar surface.
3.) There is zero or one steering link per wheel.
4.) All steering axes are perpendicular to the surface.
5.) The translational friction at the point of contact between a wheel and the surface is large enough so that no translational slip may occur.
6.) The rotational friction at the point of contact between a wheel and the surface is small enough so that rotational slip may occur.

---

4. Coordinate System Assignments

Coordinate system assignment is the first step in the kinematic modeling of a mechanism. Lower-pair mechanisms[1] (such as revolute and prismatic joints) function with two surfaces in relative motion. In contrast, the wheels of a WMR are higher-pairs; they function ideally by point contact. Because the A-Matrices which model manipulators depend upon the relative position and orientation of two successive joints, the Denavit-Hartenberg convention leads to ambiguous assignments of coordinate transformation matrices in multiple closed-link chains which are present

[1] Lower-pair mechanisms are pairs of components whose relative motions are constrained by a common surface contact; whereas, higher-pairs are constrained by point or line contact.

in WMRs. We apply the Sheth-Uicker convention to assign coordinate systems and model each wheel as a planar pair at the point of rolling contact. This convention allows the modeling of the higher-pair wheel motion and eliminates ambiguities in coordinate transformation matrices. The planar pair allows three degrees of relative motion: x and y translation, and rotation about the point-of-contact as shown in Figure 2.

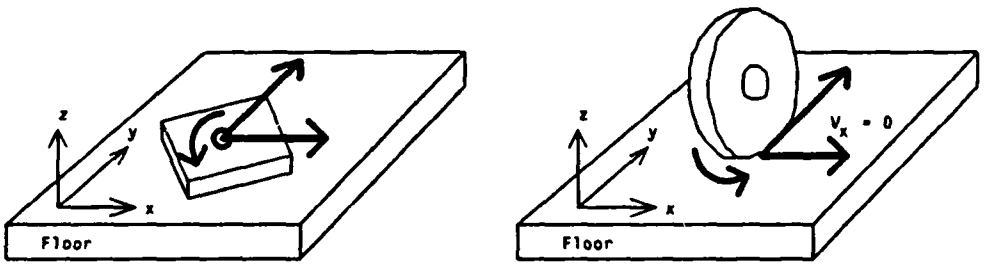


Figure 2

Planar Pair Model of a Wheel

This modeling of a WMR leads to the coordinate system assignments defined in Table 1 . The *floor* coordinate system is a reference frame for robot motions. The *robot* coordinate system is assigned to the robot body so that the position of the WMR is the relative translation from the floor coordinate system to the robot coordinate system. The *hip* coordinate system is assigned at a point on the robot body which intersects the steering axis. The *steering* coordinate system is assigned at the same point along the steering axis, but is fixed relative to the steering link. We assign a *contact point* coordinate system at the point-of-contact between each wheel and the floor.

We define an *instantaneously coincident robot* coordinate system for describing motions (i.e., velocities and accelerations) of the robot relative to its own position and orientation. We also define a function $\overline{R}(t^*)$ which returns a coordinate system that is stationary relative to the floor coordinate system and coincident with the robot coordinate system at the time $t = t^*$:

$$\overline{R}(t^*) = R\,|_{t=t^*} \quad .$$

Table 1: Coordinate System Assignments

N Number of wheels on the robot.

F *Floor* : Stationary reference coordinate system with the z-axis orthogonal to the surface of travel.

R *Robot* : Coordinate system which moves with the WMR body, with the z-axis orthogonal to the surface of travel.

H_i *Hip* (for $i = 1, ..., N$) : Coordinate system which moves with the WMR body, with the z-axis coincident with the axis of steering joint i if there is one; coincident with the contact point of coordinate system i if there is no steering joint.

S_i *Steering* (for $i = 1, ..., N$) : Coordinate system which moves with steering link i, with the z-axis coincident with the z-axis of H_i, and the origin coincident with the origin of H_i.

C_i *Contact Point* (for $i = 1, ..., N$) : Coordinate system which moves with the steering link i, with the origin at the point-of-contact between the wheel and the surface; the y-axis is parallel to the wheel (if the wheel has a preferred orientation; if not, the y-axis is arbitrarily assigned) and the x-y plane tangent to the surface.

$\overline{R}$ *Instantaneously Coincident Robot* : Coordinate system instantaneously coincident with the R coordinate system at the time t^* and stationary relative to the F coordinate system (i.e., $\overline{R}$ is the value of R at the time t^*: $\overline{R} = R\,|_{t=t^*}$).

$\overline{C_i}$ *Instantaneously Coincident Contact Point* (for $i = 1, ..., N$) : Coordinate system instantaneously coincident with the C_i coordinate system at the time t^* and stationary relative to the F coordinate system (i.e., $\overline{C_i} = C_i\,|_{t=t^*}$).

The instantaneously coincident robot coordinate system is thus a discrete sample of the continuous robot coordinate system at the time t^*. Similarly, the *instantaneously coincident contact point* coordinate system is coincident with the contact point coordinate system at the time $t = t^*$, and stationary relative to the floor coordinate system.

Placement of the coordinate systems is illustrated in Figure 3, where we show a pictorial view of a WMR. For a WMR with N wheels, we assign 4N+2 coordinate systems to the robot and one stationary reference frame.

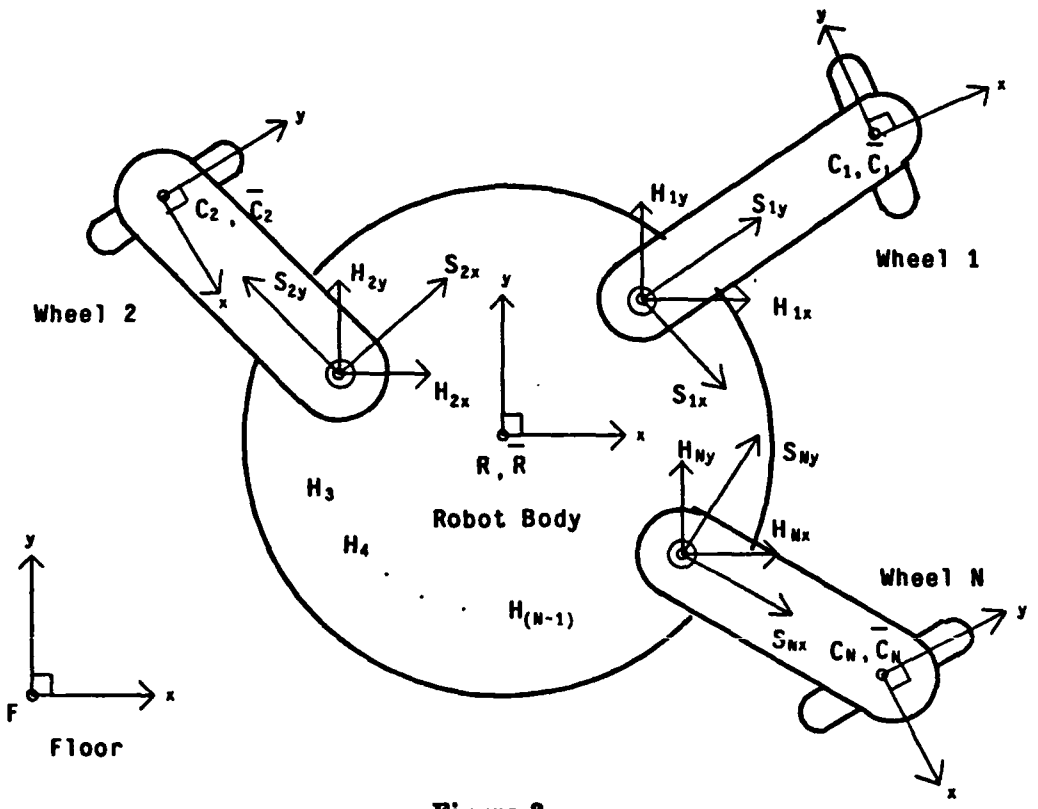


Figure 3

WMR Model Showing Placement of Coordinate Axes

5. Transformation Matrices

Homogeneous (4×4) transformation matrices are conventionally defined to express the position and orientation of one coordinate system relative to another. The transformation matrix ${}^{A}\Pi_{B}$ transforms the coordinates of point ${}^{B}\mathbf{r}$ in coordinate frame B to the corresponding coordinates ${}^{A}\mathbf{r}$ in the second coordinate frame A.

We adopt the following notation. Scalar quantities are denoted by lower case type (e.g., w). Vectors are denoted by lower case boldface type (e.g., $\mathbf{r}$). Matrices are denoted by upper case boldface (e.g., Π). Pre-superscripts denote reference coordinate systems. The pre-superscript may be omitted if the defining coordinate frame is transparent from the discussion. Post-subscripts may be used to denote coordinate systems or specific components of a vector or matrix.

Before we define the transformation matrices between the coordinate systems of our WMR

model, we define in Table 2 nomenclature for rotational and translational displacements, velocities and accelerations.

Table 2

Scalar Rotational and Translational Variables

${}^{A}\theta_B$: The rotational displacement (counterclockwise by convention) between the x-axis of the A coordinate system and the x-axis of the B coordinate system about the z-axis of the A coordinate system. ${}^{A}\dot{\theta}_B = {}^{A}v_B$ and ${}^{A}\ddot{\theta}_B = {}^{A}a_B$.

${}^{A}d_{Bj}$: (for $j \in [x, y, z]$) : The translational displacement between the origin of the A coordinate system and the origin of the B coordinate system along the j-axis of the A coordinate system. ${}^{A}\dot{d}_{Bj} = {}^{A}\omega_B$ and ${}^{A}\ddot{d}_{Bj} = {}^{A}\alpha_B$.

A transformation matrix in our WMR model embodies a rotation ${}^{A}\theta_B$ about the z-axis of coordinate system A and translations ${}^{A}d_{Bx}$, ${}^{A}d_{By}$ and ${}^{A}d_{Bz}$ along the respective coordinate axes as shown in (5.1).

$$ {}^{A}\Pi_B = \begin{pmatrix} \cos {}^{A}\theta_B & -\sin {}^{A}\theta_B & 0 & {}^{A}d_{Bx} \\ \sin {}^{A}\theta_B & \cos {}^{A}\theta_B & 0 & {}^{A}d_{By} \\ 0 & 0 & 1 & {}^{A}d_{Bz} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.1} $$

The assignment of coordinate systems results in two types of transformation matrices between coordinate systems: *constant* and *variable*. The transformation matrix between coordinate systems fixed at two different positions on the same link is constant. Transformation matrices relating the position and orientation of coordinate systems on different links include joint variables and thus are variable. Constant and variable transformation matrices are denoted by ${}^{A}\mathbf{T}_B$ and ${}^{A}\Phi_B$, respectively.

6. Matrix Coordinate Transformation Algebra

The kinematics of stationary manipulators are conventionally modeled by exploiting the properties of transformation matrices. We formalize the manipulation of transformation matrices in the presense of instantaneously coincident coordinate systems by defining *matrix coordinate transformation algebra*. An algebra consists of a set of operands and a set of operations which may be

applied to the operands. The operands of matrix coordinate transformation algebra are transformation matrices and the operations are matrix addition, multiplication, differentiation and inversion. Matrix coordinate transformation algebra allows the calculation of the relative positions, velocities and accelerations of robot coordinate systems (including instantaneously coincident coordinate systems) without physical insight. The following axioms define the special properties of transformation matrices (i.e, those properties which arbitrary matrices do not posses).

Axioms

Cascade :	${}^{A}\Pi_{C} = {}^{A}\Pi_{B}\ {}^{B}\Pi_{C}$
Inversion :	${}^{A}\Pi_{B} = {}^{B}\Pi_{A}^{-1}$
Identity :	${}^{A}\Pi_{A} = \mathbf{I}$
Instantaneous Coincidence :	$({}^{\overline{A}}\Pi_{A})\vert_{t=t^{*}} = \mathbf{I}$

The matrix *coordinate transformation axioms* lead to the following *corollaries* which we apply to the kinematic modeling of WMRs.

Corollaries

Cascade Position :	${}^{A}\Pi_{Z} = {}^{A}\Pi_{B}\ {}^{B}\Pi_{C}\ {}^{C}\Pi_{D}\ \ldots\ {}^{Y}\Pi_{Z}$
Cascade Velocity :	${}^{A}\dot{\Pi}_{Z} = {}^{A}\dot{\Pi}_{B}\ {}^{B}\Pi_{Z} + {}^{A}\Pi_{B}\ {}^{B}\dot{\Pi}_{C}\ {}^{C}\Pi_{Z} + \ldots + {}^{A}\Pi_{Y}\ {}^{Y}\dot{\Pi}_{Z}$

We make extensive use of the axioms and corollaries of matrix coordinate transformation algebra for deriving the wheel equations-of-motion.

7. Position Kinematics

We apply the transformation matrices and matrix coordinate transformation algebra to calculate the following positional kinematic relations:

1.) the position of a point r relative to one coordinate system A in terms of the position of the point relative to another coordinate system Z, and

2.) the position and orientation of a coordinate system Z relative to another coordinate system A.

Problem 1 is solved in (7.1) by applying the property of matrix transformation.

$$^{A}\mathbf{r} = {}^{A}\Pi_{Z}\ {}^{Z}\mathbf{r} \tag{7.1}$$

When the transformation matrix ${}^{A}\Pi_{Z}$ is not known directly, we apply the cascade position corollary to calculate it from known transformation matrices in (7.2).

$$^{A}\Pi_{Z} = {}^{A}\Pi_{B}\ {}^{B}\Pi_{C}\ {}^{C}\Pi_{D}\ \ldots\ {}^{Y}\Pi_{Z} \tag{7.2}$$

We must determine whether there is a complete set of known transformation matrices which can be cascaded to create the desired transformation matrix. We apply transformation graphs to resolve this problem. In Figure 3, we display a transformation graph of a WMR with one steering link per wheel.

The origin of each coordinate system is represented by a dot, and transformations between coordinate systems are depicted by directed arrows. The transformation in the direction opposing an arrow is calculated by applying the inversion axiom. Finding a cascade of transformations to calculate a desired transformation is thus equivalent to finding a path from the reference coordinate system of the desired transformation A to the destination coordinate system Z. The matrices to be cascaded are listed by traversing the path in order. Each transformation in the path which is traversed from the tail to the head of an arrow is listed as the matrix itself, while transformations traversed from the head to the tail are listed as the inverse of the matrix.

We solve problem 2 by equating components of the matrices on both sides of matrix equation (7.2), and solving for the position ${}^{A}d_{Z_x}$, ${}^{A}d_{Z_y}$ and ${}^{A}d_{Z_z}$ and the orientation ${}^{A}\theta_{Z_z}$ of coordinate system Z relative to coordinate system A.

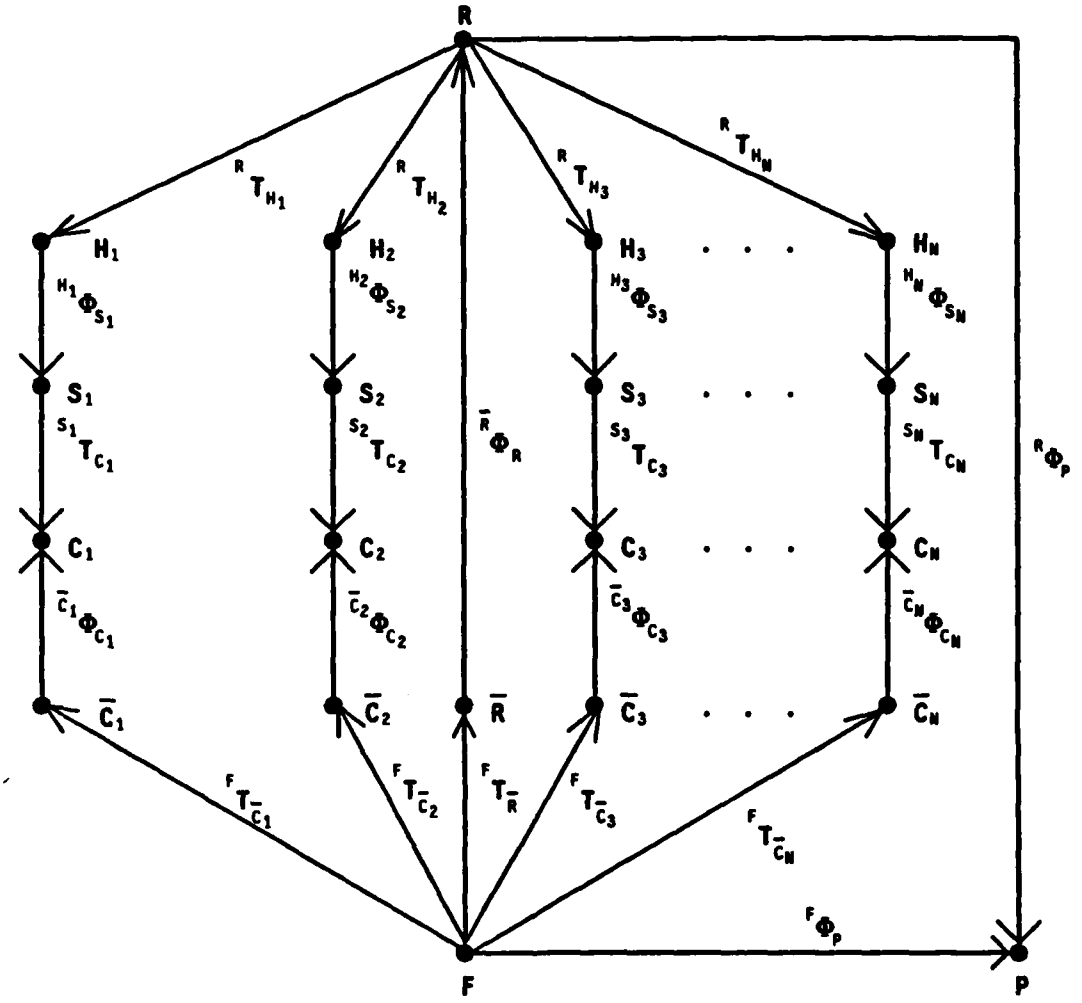


Figure 3

Transformation Graph of a WMR

8. Velocity and Acceleration Kinematics

We relate the velocities of the WMR by differentiating the position equations in Section 7. The wheel Jacobian matrix is developed by applying the cascade velocity corollary. The wheel Jacobian matrix, analogous to a manipulator Jacobian matrix, relates the component velocities of the robot ${}^{\bar{R}}v_{Rx}$, ${}^{\bar{R}}v_{Ry}$, and ${}^{\bar{R}}\omega_{Rz}$ to the velocities of the steering link ${}^{H_i}\omega_{S_i}$ and the wheel contact point ${}^{\bar{C}_i}v_{C_ix}$, ${}^{\bar{C}_i}v_{C_iy}$, and ${}^{\bar{C}_i}\omega_{C_iz}$. Some wheels do not have steering links and some do not allow motion perpendicular to the wheel orientation; thus, the number of degrees-of-freedom for wheel i is $m_i \leq 4$. The Jacobian matrix for wheel i is of dimensions $(3 \times m_i)$.

We begin development of the Jacobian matrix by applying the cascade position corollary to write a matrix equation with the unknown dependent variables (i.e., robot coordinates, ${}^{\overline{R}}\Phi_R$) on the left-hand side, and the independent variables (i.e., wheel i coordinates, ${}^{H_i}\Phi_{S_i}$ and ${}^{\overline{C_i}}\Phi_{C_i}$) on the right-hand side:

$$ {}^{\overline{R}}\Pi_R = {}^{F}\mathbf{T}_{\overline{R}}^{-1}\ {}^{F}\mathbf{T}_{\overline{C_i}}\ {}^{\overline{C_i}}\Phi_{C_i}\ {}^{S_i}\mathbf{T}_{C_i}^{-1}\ {}^{H_i}\Phi_{S_i}^{-1}\ {}^{R}\mathbf{T}_{H_i}^{-1} . \tag{8.1} $$

To introduce the velocities, we apply the cascade velocity corollary. We apply the axioms and corollaries of matrix coordinate transformation algebra to solve for the robot velocities in term of the wheel velocities:

$$ {}^{\overline{R}}\dot{\mathbf{p}} = \begin{pmatrix} {}^{\overline{R}}v_{Rx} \\ {}^{\overline{R}}v_{Ry} \\ {}^{\overline{R}}\omega_{Rz} \end{pmatrix} = \begin{pmatrix} \cos{}^{R}\theta_{C_i} & -\sin{}^{R}\theta_{C_i} & {}^{R}d_{C_iy} & -{}^{R}d_{H_iy} \\ \sin{}^{R}\theta_{C_i} & \cos{}^{R}\theta_{C_i} & -{}^{R}d_{C_iz} & {}^{R}d_{H_iz} \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} {}^{\overline{C_i}}v_{C_iz} \\ {}^{\overline{C_i}}v_{C_iy} \\ {}^{\overline{C_i}}\omega_{C_iz} \\ {}^{H_i}\omega_{S_i} \end{pmatrix} = \hat{\mathbf{J}}_i\ \dot{\hat{\mathbf{q}}}_i , \tag{8.2} $$

where $i = 1 \ldots N$ is the wheel index, ${}^{\overline{R}}\dot{\mathbf{p}}$ is the vector of robot velocities in the robot frame, $\hat{\mathbf{J}}_i$ is the pseudo-Jacobian matrix of wheel i, and $\dot{\hat{\mathbf{q}}}_i$ is the pseudo-velocity vector for wheel i. The actual velocity vector for typical wheels does not contain the four component velocities in (8.2). Typical wheels posses fewer than four degrees-of-freedom and thus fewer than four elements in the velocity vector. Further, since all actual wheel motions are rotations about physical wheel axes, the wheel velocity vector contains the angular velocities of the wheels rather than the linear velocities of the point-of-contact along the surface of travel. We relate the (4×1) pseudo-velocity vector to the $(m_i \times 1)$ actual velocity vector $\dot{\mathbf{q}}_i$ by a $(4 \times m_i)$ wheel matrix $\mathbf{W}_i$:

$$ \dot{\hat{\mathbf{q}}}_i \ = \ \mathbf{W}_i\ \dot{\mathbf{q}}_i . \tag{8.3} $$

We substitute (8.3) into (8.2) to calculate the robot velocities in terms of the wheel velocity vector in (8.4).

$$ {}^{\overline{R}}\dot{\mathbf{p}} \ = \ \hat{\mathbf{J}}_i\ \mathbf{W}_i\ \dot{\mathbf{q}}_i \tag{8.4} $$

The kinematic wheel equation-of-motion (8.4) is of the form:

$$ {}^{\overline{R}}\dot{\mathbf{p}} \ = \ \mathbf{J}_i\ \dot{\mathbf{q}}_i . \tag{8.5} $$

where $\mathbf{J}_i \ = \ \hat{\mathbf{J}}_i\mathbf{W}_i$ is the $(3 \times m_i)$ wheel Jacobian matrix for wheel i.

The accelerations of the WMR are calculated by applying the cascade acceleration corollary

to write the second derivative of the position equations in Section 7.

$$\begin{pmatrix} {}^{\overline{R}}a_{Rx} \\ {}^{\overline{R}}a_{Ry} \\ {}^{\overline{R}}\alpha_{Rz} \end{pmatrix} = \begin{pmatrix} \cos {}^{R}\theta_{C_i} & -\sin {}^{R}\theta_{C_i} & {}^{R}d_{C_iy} & -{}^{R}d_{H_iy} \\ \sin {}^{R}\theta_{C_i} & \cos {}^{R}\theta_{C_i} & -{}^{R}d_{C_ix} & {}^{R}d_{H_ix} \\ 0 & 0 & 1 & -1 \end{pmatrix} \begin{pmatrix} {}^{\overline{C_i}}a_{C_ix} \\ {}^{\overline{C_i}}a_{C_iy} \\ {}^{\overline{C_i}}\alpha_{C_i} \\ {}^{H_i}\alpha_{S_i} \end{pmatrix} \tag{8.6}$$

$$+ \begin{pmatrix} {}^{R}d_{C_ix} & {}^{R}d_{H_ix} & {}^{R}d_{H_ix} \\ {}^{R}d_{C_iy} & {}^{R}d_{H_iy} & {}^{R}d_{H_iy} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} {}^{\overline{C_i}}\omega_{C_i}^2 \\ -2\,{}^{\overline{C_i}}\omega_{C_i}\,{}^{H_i}\omega_{S_i} \\ {}^{H_i}\omega_{S_i}^2 \end{pmatrix}$$

The robot accelerations in (8.6) are composed of three acceleration components: the wheel accelerations (${}^{\overline{C_i}}a_{C_ix}$, ${}^{\overline{C_i}}a_{C_iy}$ and ${}^{\overline{C_i}}\alpha_{C_i}$); the *centripetal* accelerations (${}^{\overline{C_i}}\omega_{C_i}^2$ and ${}^{H_i}\omega_{S_i}^2$) having squared velocties; and the *coriolis* accelerations (${}^{\overline{C_i}}\omega_{C_i}\,{}^{H_i}\omega_{S_i}$) having products of different velocities.

9. The Composite Robot Equation-of-Motion

We combine the equations-of-motion of each wheel on a WMR to form the composite robot equation. Two solutions of the composite robot equation have practical applications. The inverse solution computes the actuated wheel velocities in terms of the robot velocity vector. The forward solution is the least-squares solution of the robot velocity vector in terms of the sensed wheel velocities.

The inverse solution is calculated independently for each wheel on a WMR by applying the inverse Jacobian matrix. The actuated velocities are extracted from the solution for application to robot control.

The least-squares forward solution provides the optimal solution of the robot velocities in the presense of sensor noise and wheel slippage in the sense that the sum of the squared errors in the velocity components is minimized. We may insure that the solution can be calculated by proper WMR design. We find that the forward solution may be simplified by eliminating the equations-of-motion of any wheel having three non-sensed degrees-of-freedom (e.g., a castor) because they do not change the solution.

A study of the nature of the solutions of the composite robot equation illuminates robot motion, actuation and sensing characteristics. Of particular importance are the conditions under which actuation of a set of the wheel degrees-of-freedom causes undesirable overdetermined and undetermined solutions. We prefer determined actuation structures because they allow control over

all robot degrees-of-freedom and do not cause undesirable actuator conflict. We also propose that overdetermined sensing structures are preferable because the least-squares forward solution tends to reduce the effects of sensor noise with redundant measurements.

We calculate the inverse and forward solutions by applying the kinematic equations-of-motion of each wheel in three dimensions x, y, and θ. If a WMR is constrained by the wheel arrangement to move in only two dimensions, we may calculate the inverse and forward solutions in an analogous manner by eliminating the third dimension from the wheel Jacobian matrices.

10. Applications

The kinematics of WMRs play important roles in modeling, design and control. We introduce five practical applications of our kinematic methodology in this section. We apply the results of our study of the composite robot equation-of-motion to the *design* of WMRs. WMRs can be designed to satisfy desirable mobility characteristics such as two and three degrees-of-freedom and the ability to actuate and sense the degrees-of-freedom. *Dead reckoning* is the real time integration of the robot velocity calculated from wheel sensor measurements. Kinematics-based WMR control systems are implemented by applying the inverse solution in the feedforward path and dead reckoning in the feedback path such that the error between the actual robot position and desired robot position is continually reduced. An improved controller is possible by applying knowledge of the robot dynamics. Our kinematic methodology is the foundation of *dynamic modeling* of WMRs. Accurate robot *control* systems rely on both kinematic and dynamic models. We also apply the kinematic equations-of-motion to the *detection of wheel slip*. When a WMR detects the onset of wheel slip the current robot position is corrected by utilizing slower absolute locating methods such as computer vision before continuing motion. The control system is thus able to track desired trajectories more accurately by continually insuring an accurate measure of robot position.

11. Summary of Kinematic Modeling Procedure

We have formulated a systematic procedure for modeling the kinematics of a WMR. In this section we summarize the modeling procedure to outline a step-by-step enumeration of the methodology to facilitate engineering applications.

1.) **Make a sketch of the WMR.** Show the relative positioning of the wheels and the steering links. The sketch need not be to scale. A top and a side view are typically sufficient.

2.) **Assign the coordinate systems.** The robot, hip, steering, contact point and floor

coordinate systems are assigned according to the conventions introduced in Table 1.

3.) **Assign the (4 × 4) coordinate transformation matrices.** The robot-hip, hip-steering, and steering-contact transformation matrices are assigned as described in Section 5.

4.) **Formulate the wheel equations-of-motion.** The position, velocity and acceleration wheel equations-of-motion are developed by applying transformation graphs and matrix coordinate transformation algebra. The specific equations required will depend upon the application.

5.) **Formulate the composite robot equation-of-motion.** The individual wheel equations are combine to model the motion of the robot.

6.) **Solve the composite robot equation.** The inverse solution and the forward solution may be calculated depending on the application.

The reader is refered to the full paper for further details.

12. Continuing Research

Our study of wheeled mobile robots is motivated by the need for designing robust feedback control algorithms for their accurate motion control. We are proceeding by paralleling the development of robust dynamic *manipulator* control systems. The first step, that of developing a kinematic model, is documented in this paper[1]. We are applying the kinematic model to develop dynamic models of WMRs. The composite kinematic-dynamic WMR model will lay the foundation for WMR control. We will apply the robot models in simulation to facilitate the design of control systems. The performance of candidate WMR control systems will be evaluated in simulation prior to time-consuming hardware implementation. In parallel with our engineering activities, we are implementing a practical control system for the newly constructed WMR *Uranus*. The theoretical and practical studies are proceeding concurrently, each reinforcing the the results of the other.

[1] P. F. Muir and C. P. Neuman, "Kinematic Modeling of Wheeled Mobile Robots," Technical Report, The Robotics Institute, Carnegie-Mellon University, Schenley Park, Pittsburgh, PA. 15213, January 1986.

Feasibility of Dynamic Trajectories for Mobile Robots

Dong Hun Shin

Department of Mechanical Engineering
Carnegie – Mellon University
Pittsburgh, PA 15213
November 1985

Abstract

Constraints for the feasible dynamic trajectories of the mobile robot are considered and conditions on the slippage between wheels and terrain are presented for testing the feasibility of dynamic trajectories. Slippage constraints are devided into two cases, the translational slippage and the loss of the traction and each case is investigated using newtonian mechanics and coulomb's friction law.

1 Introduction

This paper concerns the feasibility of the dynamic trajectories used for the supervisory steering control of the wheeled mobile robots. The steering control objective is to navigate the robot among obstacles to reach the specified destination. A usual steering control problem of a mobile robot consists of three hierachical structures [4] [5] [1] which are illustrated in Fig 1.

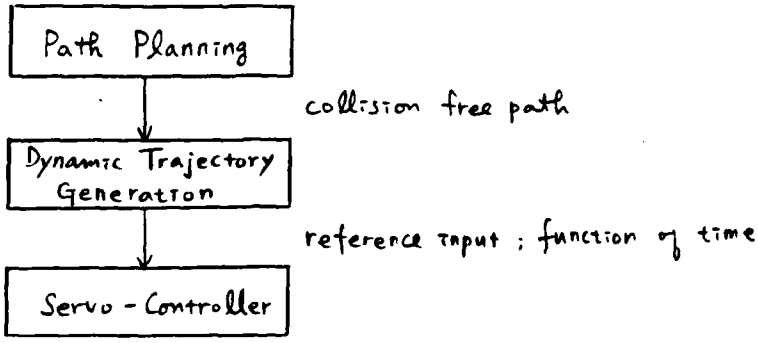


Fig 1 hierachical structure of the supervisory steering control

The first level of the control hierachy is to plan a collision free path which is usually a sequence of nodes from the current positin to the destination. A dynamic trajectory is then generated which takes into consideration system dynamics and limits on control inputs. This trajectory is converted into reference control trajectories for the servo-controlled actuator inputs.

The issues addressed in this paper is the feasibility condition of the dynamic trajectories of the mobile robot. Since the feasibility of the trajectories depend on the constraints of the control system, constraints of the mobile robot are discussed and especially, slippage constraints which are the crucial and characteristic constraints for the feasible trajectories of the mobile robot are investigated.

The remainder of this paper is organized as follows. The feasibility problem of the trajectory is formulated in section 2. The potential sources of infeasibility are discussed in section 3. Section 4 presents the feasibility condition due to slippage constraints. The concluding section identifies several directions for future research.

2 Problem Formulation

The dynamics of a mobile robot with n degrees of freedom can be represented by n coupled second order differential equation (1).

$$f(q,\dot{q},\ddot{q},\tau) = 0 \qquad (1)$$

where

$i=1,\ldots,n$
$q=[q_1,\ldots,q_n]$ · n generalized coordinates
$\dot{q}=[\dot{q}_1,\ldots,\dot{q}_n]$ · n generalized velocities
$\ddot{q}=[\ddot{q}_1,\ldots,\ddot{q}_n]$ · n generalized acceleration
$\tau=[\tau_1,\ldots,\tau_n]$ · n generalized forces

If we let Q be the $2n$ dimensional set of feasible generalized coordinates and velocities, physical operating region of system is expressed as

$$(q,\dot{q}) \in Q \tag{2}$$

Since the generalized forces are combination of components of control input forces/torques, they are also limited as

$$\tau_i^- \leq \tau \leq \tau_i^+ \tag{3}$$

where, $i=1,\ldots,n$

The task of the mobile robot is normally specified in the global coordinate frame where the destination and the obstacles can be most easily represented. Thus computation of the steering control in terms of the generlaized coordinates requires mapping the destination and the obstacles into the generalized coordinate frame and solving a nonlinear control problem with state variable constraints. But it is not easy. A tractable approach to steering control is to plan the collision free path in the global coordinate frame independently of the dynamic constraints. A dynamic trajectory is then generated in global coordinates as a function of time with respect to the specific point of the robot.

$$X=X(t) \tag{4}$$

where, $X=[X_1,\ldots,X_n]$: trajectory in global frame

These dynamic trajectory are then converted into the generalized coordinates as reference input for the lower level servo controller.

$$q^t = q^t(t) \tag{5}$$

where

$i=1,\ldots,n$
q^t : trajectory in generalized frame

These dynamic trajectories must satisfy the equation of motion under the constraints (2) and (3).

As an example, we consider a simplified model of a tricycle which moves on a planar surface and is configured in Fig 2. It goes only forward and has one steered and driven front wheel and two rear idle wheels with same radius.

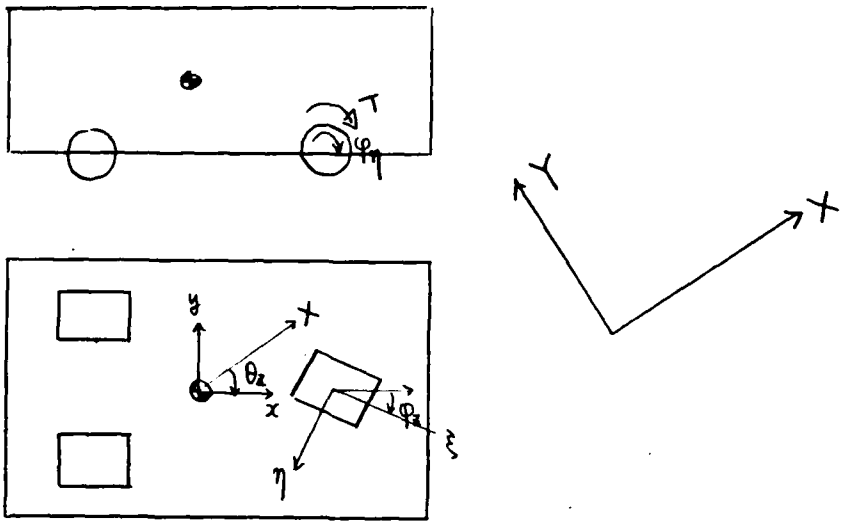


Fig 2. a simple model of Tricycle

where,

X,Y,z are the inertial global coordinates

x,y,z are the body coordinates which is fixed to the mass center of the robot and translates with velocity V_g and yaws with angular velocity θ_z with respect to the inertial coordinate frame.

φ_z : steering angle of the driven wheel

φ_η : rolling angle of the driven wheel

T_z : torque to steer the wheel

T_η : torque to turn the wheel

If we consider the degrees of freedom for the tricycle model, the three coordinates X,Y and θ_z constitute a complete set to express the position and the orientation of the robot. The variation dX,dY and $d\theta_z$ are not, however, independent, since the requirment that any translation must be in the heading direction implies the constraining relation .

$$\frac{dY}{dX}=\tan\theta_z$$

In other words, there is one nonholonomic constraint. Thus the degrees of freedom of the tricycle model for the planar motion is two, which is known as the minimum degree of freedom for the two dimensional planar motiojn [6], as the conventional steered vehicle has two degrees of freedom.

Then, two generalized coordinates and forces for the tricycle model can be taken as

$$q=[\varphi_\eta, \varphi_z]$$
$$\tau=[T_\eta, T_z]$$

The simple operating region of the tricycle can be represented as

$$0 \le \varphi_\eta \cdot |\varphi| \le \varphi_{z,max} \qquad (6)$$
$$\varphi_\eta \le \varphi_{\eta,max} \cdot |\varphi_z| \le \varphi_{z,max}$$

And the limit on control inputs can be specified as

$$0 \le T_\eta \le T_{\eta,max}$$
$$|T_z| \le T_{z,max}$$

The dynamic trajectory with respect to the mass center of the tricycle can be generated in the inertial coordinate frame as

$$X = f_X(t)$$
$$Y = f_Y(t)$$

and these trajectories can be converted into the generalized coordinates as

$$\varphi_\eta = f_\eta(t)$$
$$\varphi_z = f_z(t)$$

3 Potential Sources of Infeasibility

This section gives a brief discussions of each major potential source of the infeasibility of the dynamic trajectory. Major potential sources are as follows.

1. Kinematic constraints
2. Vehicle stability
3. Limits on control input force/torque
4. Slippage of wheels

The first, kinematic constraints, can be thought as equation (2) or (6), feasible generalized

coordinates and velocities. This is one of the major constraints problem for the steering control of manipulator because it limits the working space and velocities in terms of the generalized coordinates and constraints are coupled with each generalized coordinates and velocities. But, generally there is no significant problem to deal with these constraints of the mobile robot because they are not coupled seriously like the constraints of the tricycle model (6). The overturn of the mobile robot during turning around or acceleration would be thought as another constraint from the view point of the vehicle stability. This constraint depends on the height of mass center, geometric composition of wheels, angular velocity and acceleration, etc.

The control input forces/torques are limited by the servo motor which is specified in the local generalized coordinate frame as equation (3). If the input forces/torques required by the trajectory (4) or (5) exceed the limit on control input, the trajectory will not be feasible. Control input constraint problem is very important to enable the robotic manipulators to perform their maximum capability and efficiency, which lead to high productivity. So the industrial manipulator control problem against these constraints has been the issue and trajectories even optimized with respect to time and energy was reported [2] [3].

Last, slippage constraints are the characteristics of the mobile robot problem. A wheel rolls due to the driven torque and frictional force between the wheel and terrain. If the actual frictional force is not sufficient, the wheel will slip. Thus the slippage constraint of a wheel is expressed as (using Coulomb's friction law)

$$F \leq \mu N \tag{7}$$

where, F : frictional force
$g(m)$: friction coefficient
N : normal force

If the wheels of the mobile robot slip, the robot will slip and leave the given dynamic trajectory, that is, the trajectory will not be feasible. Thus the dynamic trajectory must be constrained to guarantee no slip of the wheel of the robot. Slippage constraint problem is thought as the most important for the feasible dynamic trajectories of the mobile robots because of the following reasons.

1. Kinematic constraints are the most. crucial for the feasible trajectories but generally can be represented easily because they are not coupled seriously in the mobile robot problem. Also it can be easily checked.

2. Vehicle stability constraints, i.e. the overturn of the robot, would not be serious, if it is taken care of at the design of the robot. Then, slippage of wheels will occur before a overturn as the conventional vehicle does.

Next section will present a approach to deal with the slippage constraints.

4 Slippage Constraints

It is difficult to solve the general slippage constraint problem and to obtain the required frictional force of each wheel for the feasible dynamic trajectory. If the trajectory is feasible, there is no slip at the point of contact between any wheel and terrain. In other words, the point of contact is momentarily at rest. Then, since no work is done by the frictional forces, there is no explicit term of the frictional forces in the equation of the motion (1). Hence, the frictional forces can not be computed with the equation of the motion (1) and the given trajectory (4) or (5). Those forces would be obtained complicatedly with the geometric constraints of the robot and the equations of the motion of the subsystems. To make the problem tractable, slippage constraints are divided into two cases under the following assumptions.

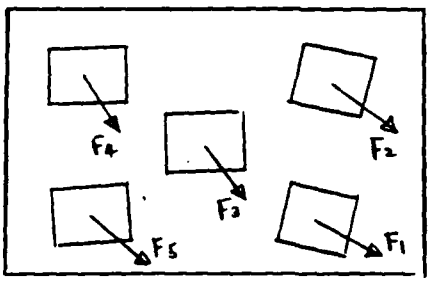
1. The robot does not have any flexible part.

2. The robot moves on a planar surface with no irregulities.

3. The frictional coefficient, μ, is constant.

4.1 Translational Slippage

We first consider the translational slippage of the wheel when there is no slip due to the loss of traction. A general m wheeled mobile robot with the frictional forces required by the trajectory are simply configured in Fig 3.



$F_1, \cdots F_5$: frictional force

Fig 3 m wheeled mobile robot and friction

Since all the wheels are fixed to the robot, it can be thought as a rigid body under external force F_j $j=1,\ldots,m$, and driven torques. Thus any part of a rigid body can not slip unless the whole body, or mass center, slips. So, we make assumption 4 as

4. Any wheel do not slip translationally unless the mass center of the robot slips.

In other words, the required frictional force of any wheel, F_j , will not exceed μN_j unless the total sum of F_j exceeds the total sum of μN_j If there is no translation slip, the next equation should be satisfied.

$$\sum_{j=1}^{m} F_j \leq \sum_{j=1}^{m} \mu N_j \tag{8}$$

Since the positions and orientations of the mass center of the robot in the *body coordinate frame* can be computed from by the dynamic trajectory, we obtain velocities of the mass center

$$V_x(t), \quad V_y(t), \quad \dot{\theta}_z(t)$$

And we can obtain relations from the equations of the motion of the m wheeled mobile robot

$$\sum_{j=1}^{m} F_{jx} = M(\dot{V}_x - V_y\dot{\theta}_z)$$

$$\sum_{j=1}^{m} F_{jy} = M(\dot{V}_y - V_x\dot{\theta}_z)$$

$$\sum_{j=1}^{m} N_j - Mg = 0$$

where M is the mass of the robot.
g is the gravitational constant.

Then, the equation (8) becomes

$$[(\dot{V}_x - V_y\dot{\theta}_z)^2 + (\dot{V}_y + V_x\dot{\theta}_z)^2]^{\frac{1}{2}} \leq \mu g \tag{9}$$

As examples, we consider simple circular motions of the tricycle model in section 2. First, if a circular motion with constant angular velocity, $\dot{\theta}_z = \omega$, is considered as in Fig 4

Then,

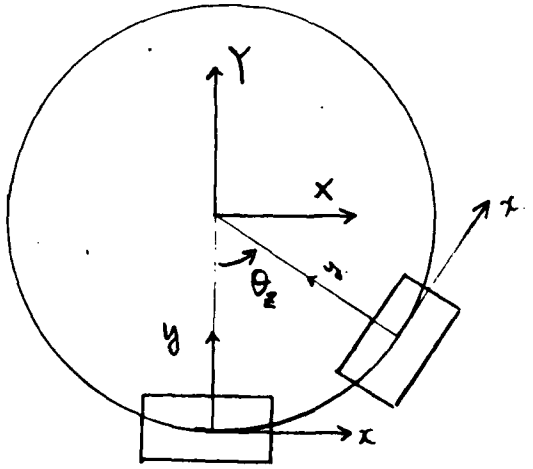


Fig 4 simple circular motion of tricycle

$$V_x = \omega R \quad V_y = 0$$
$$\dot{V}_x = 0 \quad \dot{V}_y = 0$$

We obtain th relation from equation (9)

$$\omega^2 R \leq \mu g$$

which agrees with the physical understanding.

Second, if a circular motion with constant angular acceleration, $\theta_z = \alpha t$, is considered, then

$$V_x = \alpha t R \quad V_y = 0$$
$$\dot{V}_x = \alpha R \quad \dot{V}_y = 0$$

And we obtain the relation from equation (9)

$$t \leq \left[\frac{\mu^2 g^2}{\alpha^4 R^2} - \frac{1}{\alpha^2}\right]^{\frac{1}{2}}$$

4.2 Loss of Traction

Next, we consider the slippage of one driven wheel due to the loss of traction. A simplified driven wheel under torque T is figured.

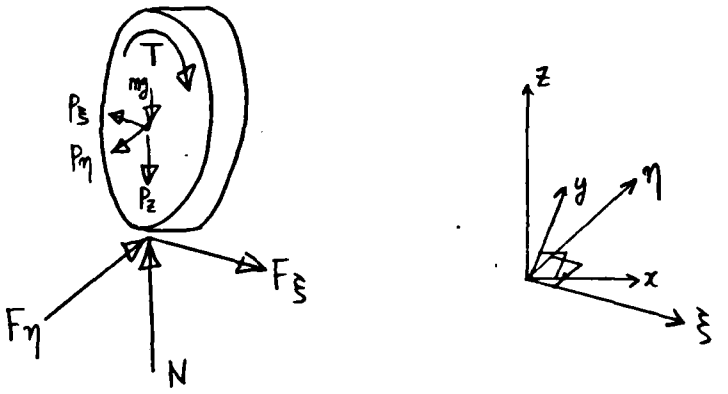


Fig 5 Driven wheel and external forces

The frictional force required by the trajectory can be decomposed into the longitudinal and the traverse force with repect to a wheel; F_ξ, F_η and from equation (7), the wheel slips if next equation is not satisfied.

$$F = [F_\xi^2 + F_\eta^2]^{\frac{1}{2}} \leq \mu N \qquad (10)$$

As the driven torque is increased for the ecceleration, the frictional force F will be increased and eventually become the maximum feasible frictional force μN. Then the torque is increased more and the wheel, however, does not slip until F_ξ alone exceeds the μN if the assumption 4 holds: there is no movement of the wheel due to slippage without the slippage of the whole robot. Physically,as F_ξ is increased, F_η will be decreased while F is μN. Thus if there is no translational slippage of the robot, any wheel does not slip provided

$$F_\xi \leq \mu N \tag{11}$$

From the Fig 5, the equation of the motion is

$$I_\eta \theta_\eta = T - F_\xi r$$

Then, equation (11) becomes

$$\frac{T - I_\eta \theta_\eta}{r} \leq \mu N \tag{12}$$

So, the mobile robot will not slip as long as equation (9) and (12) hold.

5 Conclusion and Future Research

In this paper, we have discussed the constraints for the feasible dynamic trajectory and presented an approach for a slippage constraints which are the most important and characteristic contraints to the mobile robot dynamic trajectory. Directions for future research include

- Derivation of constraints on vehicle stability.
- Methods for the generation of the feasible dynamic trajectory considering constraints discussed in section 3.
- Implementation of feasibility constraints to the dynamic steering control of the mobile robot.
- Modification of the dynamic steering algorithm so that it may be applied to the navigation of the current mobile robot.
- Integration of the dynamic steering algorithm with higher level planning or previous information.
- Navigation of the mobile robot using the dynamic steering control.

References

[1] G. Giralt, R. Chatila, M. Vaisset.
An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots.
In *1st International Symposium of Robotic Research*. Bretton Woods, NH, September, 1983.

[2] B. H. Krogh, T. J. Graettinger.
Manerverability Constraints for Supervisory Steering Control.
In *24th Conference on Decision and Control*. Fort Lauderdale, FL, Dec, 1985.

[3] B. K. Kim and K. G. Shin.
Suboptimal Control of Industrial Manipulators with a Weighted Minimum Time-Fuel Criterion.
IEEE Trans. on Automatic Control 30(1):1-10, Jan, 1985.

[4] E. Koch, C. Yeh, G. Hillel, A. Meystel, C. Isik.
Simulation of Path Planning for a System with Vision and Map Updating.
In *1985 IEEE International Conference on Robotics and Automation*. Mar, 1985.

[5] B.H. Krogh.
A Generalized Potential Field Approach to Obstacle Avoidance Control.
In *Robotics Research: The Next Five Years and Beyond*. Bethlehem, PA, Aug, 1984.

[6] K. J. Waldron.
Mobility and Controllability Characteristics of Mobile Robotic Platform.
In *1985 IEEE International Conference on Robotics and Automation*. Mar, 1985.

The Robots

The NEPTUNE Mobile Robot

Gregg W. Podnar
Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

Neptune is a functional vehicle for autonomous mobile robot research. As a reliable mobile base, it supports experiments in perception, real-world modeling, navigation, planning and high-level control. It is self-propelled, with computer control of direction and motivation.

One of the prime design goals was the minimization of the number of subsystems. By doing so, reliability was enhanced.

Structure

Neptune's basic structure is best likened to a child's tricycle. The three 10-inch (25cm) pneumatic tires are used to provide spring, compliance, and traction on soft ground.

Steering of the fork is accomplished by one motor. The fork-mounted wheel is driven by a second motor. This allows sharp turning which facilitates navigation in cluttered environments. The other two wheels are parallel and rotate freely. The fork can turn at least 90° left and right, and the wheel can be driven forward or back. Together, these two features enable the vehicle to rotate about a vertical axis through a point located directly between the two passive wheels. The overall width is 22.5 inches (57cm), and the length is 32.5 inches (83cm). The turning length 'curb-to-curb' is only 42 inches (107cm).

Power

To eliminate on-board power storage and recharging, mains power is supplied through an umbilical. This 120VAC is distributed for all on-board electrical equipment via outlets mounted in the vehicle frame. Each piece of equipment provides its own power conversion/protection.

Motors

Using 120VAC motors eliminates the need for massive power conversion equipment. Synchronous motors were chosen for drive and steering as this replaces a feed-back and servoing system (Run a motor for a length of time, and *calculate* the revolutions.). The elimination of optical encoders or resolvers enhances reliability.

Control

An on-board processor accepts commands from a serial data link through the umbilical. This processor controls the motor relays and monitors fork position. It also provides control and monitoring for other vehicle-mounted equipment (such as switching between two television cameras).

Communication

Together with the Power, the umbilical carries cables for digital and video signals to and from off-board computers.

Construction

Neptune is made from two basic assemblies, the **Fork** and the **Frame**. Both parts were designed to have an excess of structural fortitude to withstand abuse and provide secure mounting points for auxiliary experimental equipment.

The frame is made of four pieces of four inch square aluminum tubing which are are welded together. Likewise, the four major fork pieces are aluminum and are welded. This was done mainly for strength but it also reduced the required machining. Once all the pieces were made, assembly of the mechanical parts took less than a week.

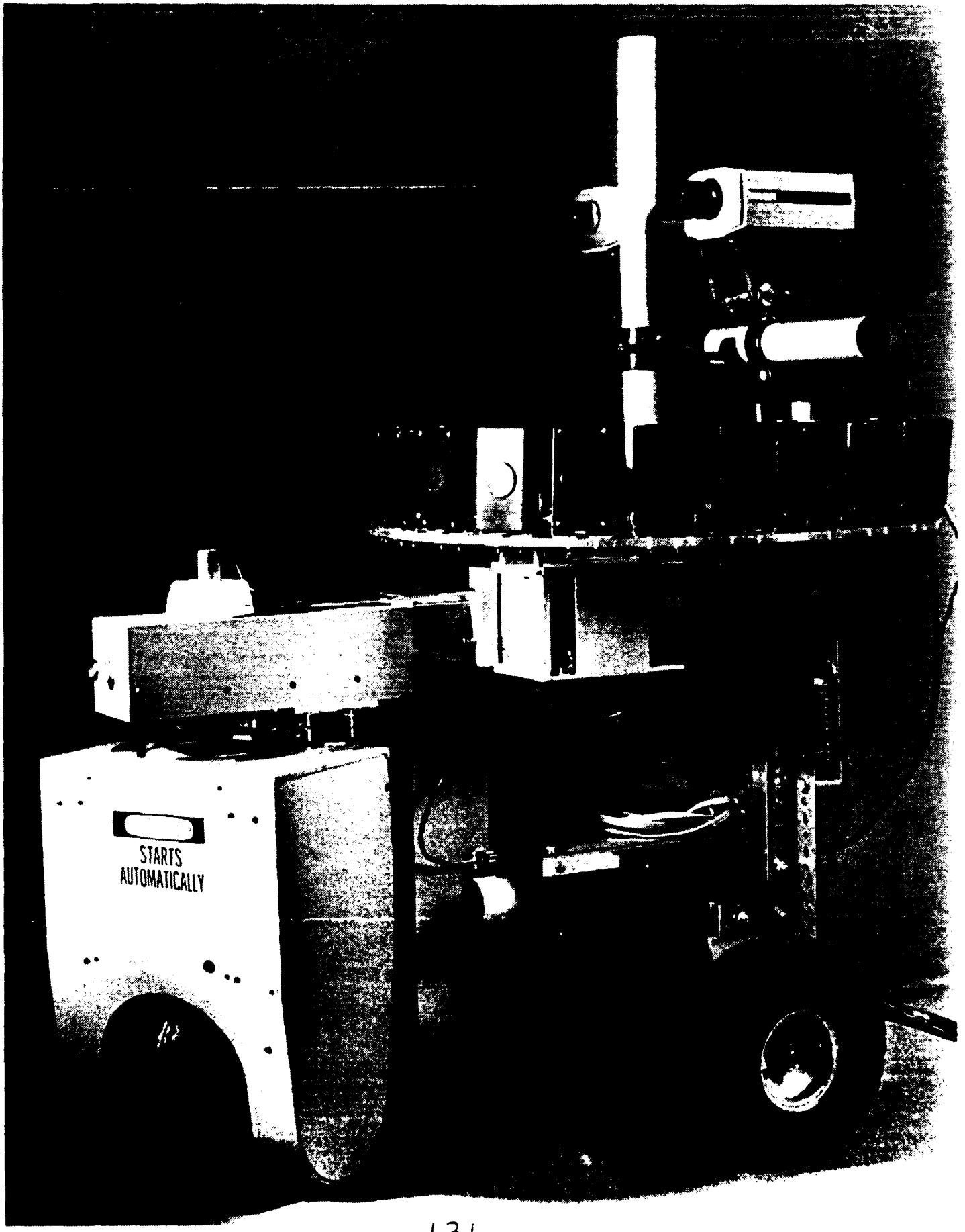
Prefabricated Components

For mounting the rotating shafts (two axles and the fork neck), off-the-shelf, housed bearings are used. In the same way, the chains and sprockets for driving and steering are standard components. The wheels and tires are units manufactured for handtrucks. Delivery time on these items is short, on the order of one to three weeks. By employing pre-fabricated components, shop time was minimized. It took one machinist about one full week to make all the other parts.

Performance

The Drive motor provides 1800 oz.in. of torque. With the 4:1 reduction gearing, about 90 pounds of pull is developed at the drive wheel. Fully loaded with cameras and a ring of 24 sonar sensors, *Neptune* weighs about 200 pounds and easily manages a 10° slope. It travels at about nine inches per second; about 1/2 MPH.

Neptune has had different configurations of sensor systems mounted on it to perform a variety of experiments. It has navigated in hallways, cluttered labs and sidewalks. It was even used in the rain with the addition of an umbrella to protect the electronics. It has reliably served our research purposes since early in 1984.


STARTS
AUTOMATICALLY

The URANUS Mobile Robot

Gregg W. Podnar
Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

Uranus is a sophisticated vehicle for autonomous mobile robot research. As an omni-directional mobile base, it makes possible experiments in perception, real-world modeling, navigation, planning and high-level control. It is self-propelled and can support a wide variety of sensor and manipulator packages. True autonomy is possible as electrical and computing power are carried on-board.

The most unique feature of *Uranus* is its four wheels. Developed by a Swedish company, MECANUM, for omni-directional movement of factory floor pallets and wheel chairs, we have adapted them for use in mobile robots. With respect to the wheels' Swedish origin, we pronounce *Uranus*: Oo-ron'-oos.

Wheels

Each wheel has twelve free-spinning rubber rollers around its circumference. The axle of each roller is at a 45° angle to a line parallel to the wheel's axle. When viewed from the side, the end of each roller overlaps the beginning of the next, and due to the barrel shape of each roller, the wheel presents a circular silhouette. As a wheel rolls, its contact with the ground changes from one roller to the next smoothly.

There are right-handed and left-handed wheels which can be thought of as working in pairs, with each pair on a common axis. When both wheels are rotated in the same direction, the sideways components generated by the rollers cancels and the wheels move forward or back. However, when the wheels are rotated in opposite directions, the sideways components add and the wheels move sideways.

Structure

Uranus describes a rectangular envelope which is 30" (76cm) long by 24" (61cm) wide by 12"(30cm) high, with additional height of 0.5"-2.5" (1.3-6.3cm) due to ground clearance. The primary frame components are 3"x6" (7.6x15.2cm) rectangular aluminum tubing. The suspension components are all stainless steel.

The vehicle has three layers. The first six inches (15cm) includes the wheels, drivetrain, motors, batteries and power control. As this is the majority of the weight, the center of gravity is very low.

The second six inches (15cm) includes computers and control electronics along with their associated power supplies. The four corners of this level are for springs and dampers of the suspension.

The third level consists of the top plate or deck. It is 23" (58cm) by 27" (69cm); slightly smaller than the vehicle envelope. This allows the wheels to contact a vertical obstacle first. The deck provides structural support for up to 250 pounds (113kg) of additional equipment. It is full of 1/4"-20 holes on a grid of one inch (2.5cm) centers.

Motors

Each of the four wheels is driven by a samarium-cobalt brushless D C. motor. An on-board computer controls motor position, speed and rotation by monitoring shaft position with an optical encoder. The motors are mounted in the side frame pieces of the first layer between the wheels. The shaft end of the motor protrudes into the frame and connects with the drivetrain. The power electronics for switching a motor's coils is housed in a heat sink mounted directly to the outboard side of the motor housing. This is to minimize EMI and allow convection cooling.

Suspension

Each wheel is mounted on what can most easily be described as a trailing-arm. Vertical movement of two inches (5cm) maximum is possible. Initially, the vehicle is suspended on stiff coil springs which allow just enough compliance to ensure that all four wheels have adequate contact with the ground. Space is available for the option of an active suspension. By computer control of pneumatic or hydraulic actuators, the vehicle can be leveled, raised and lowered to facilitate certain environments.

Power

Power is supplied by an on-board sealed lead-acid battery. The motors operate directly from the 24VDC battery power, whereas the computers and other equipment convert and condition power through dedicated switching power supplies.

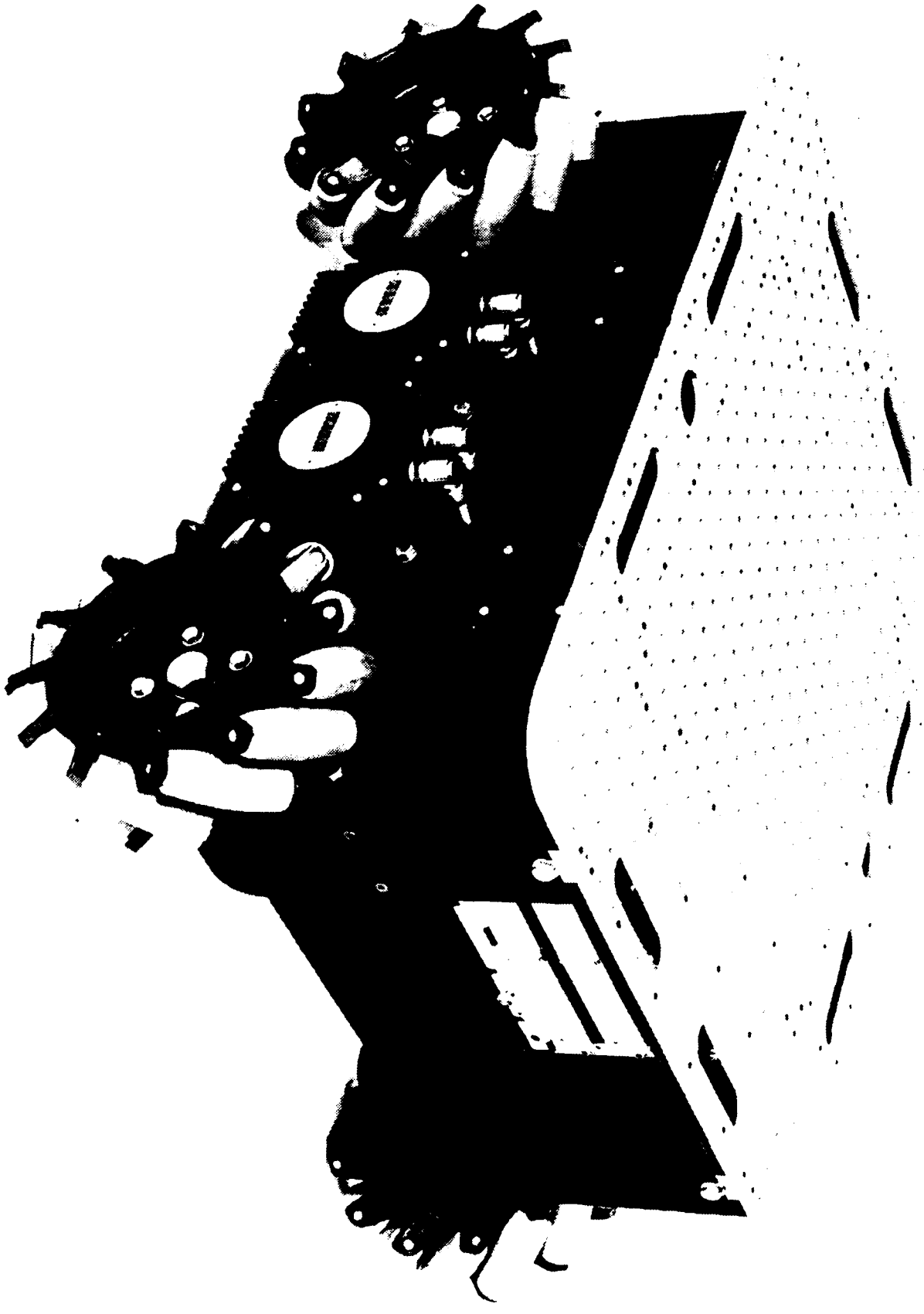
An umbillical provides 24VDC from an off-board supply. This supply is capable of powering the entire vehicle and simultaneously charging the batteries. In this way, experimentation which does not require full wireless operation and indefinite operating times are facilitated.

Performance

Four motors, developing peak torque of 3.5 ft.lbs. (4.7nm) drive the wheels through a 4:1 reduction. With a 9" (23cm) wheel diameter, about 150 lbs. (660nt) of thrust is developed. This is the theoretical maximum; about half this number is a practical value.

With these motors the maximum speed is about three feet (1m) per second or 2MPH (3.2KPH) which is adequately fast for a cluttered environment. This can be increased if need be.

With on-board batteries, about four hours of wireless operation is possible. This estimate must be reduced if the vehicle requires more power for rough terrain or interaction with objects in the environment. Similarly, more time is available for a single experiment if the movements are more sedate.



Motivation

Robots That Rove

Hans P. Moravec
Robotics Institute
Carnegie-Mellon University
Pittsburgh, PA 15213

August, 1985

The most consistently interesting stories are those about journeys, and the most fascinating organisms are those that move from place to place. I think these observations are more than idiosyncrasies of human psychology, but illustrate a fundamental principle. The world at large has great diversity, and a traveller constantly encounters novel circumstances, and is consequently challenged to respond in new ways. Organisms and mechanisms do not exist in isolation, but are systems with their environments, and those on the prowl in general have a richer environment than those rooted to one place.

Mobility supplies danger along with excitement. Inappropriate actions or lack of well-timed appropriate ones can result in the demise of a free roamer, say over the edge of a cliff, far more easily than of a stationary entity for whom particular actions are more likely to have fixed effects.

Challenge combines with opportunity in a strong selection pressure that drives an evolving species that happens to find itself in a mobile way of life in certain directions, directions quite different from those of stationary organisms. The last billion years on the surface of the earth has seen a grand experiment exploring these pressures. Besides the fortunate consequence of our own existence, some universals are apparent from the results to date and from the record. In particular, intelligence seems to follow from mobility.

I believe the same pressures are at work in the technological evolution of robots, and that, by analogy, mobile robots are the most likely route to solutions to some of the most vexing unsolved problems on the way to true artificial intelligence - problems such as how to program common sense reasoning and learning from sensory experience. This opportunity carries a price - programs to control mobile robots are more difficult to get right than most - the robot is free to search the diverse world looking for just the combination that will mess up your plan. There's still a long way to go, but perhaps my experiences thus far pursuing this line of thought will convince you as they have me. Among the conclusions that surprised me is that future intelligent robots will of necessity be more like animals and humans that I used to believe, for instance they will exhibit recognizable emotions and human irrationalities. On to cases.

Mobility and Intelligence in Nature

Two billion years ago our unicelled ancestors parted genetic company with the plants. By accident of energetics and heritage, large plants now live their lives fixed in place. Awesomely effective in their own right, the plants have no apparent inclinations towards intelligence; a piece of negative evidence that supports my thesis that mobility is a parent of this trait.

Animals bolster the argument on the positive side, except for the immobile minority like sponges and clams that support it on the negative.

A billion years ago, before brains or eyes were invented, when the most complicated animals were something like hydras, double layers of cells with a primitive nerve net, our progenitors split with the invertebrates. Now both clans have intelligent members. Cephalopods are the most intellectual invertebrates. Most mollusks are sessile shellfish, but octopus and squid are highly mobile, with big brains and excellent eyes. Evolved independently of us, they are different. The optic nerve connects to the back of the retina, so there is no blind spot. The brain is annular, a ring around the esophagus. The green blood is circulated by a systemic heart oxygenating the tissues and two gill hearts moving depleted blood. Hemocyanin, a copper doped protein related to hemoglobin and chlorophyll, carries the oxygen.

Octopus and their relatives are swimming light shows, their surfaces covered by a million individually controlled color changing cells. A cuttlefish placed on a checkerboard can imitate the pattern, a fleeing octopus can make deceiving seaweed shapes coruscate backward along its body. Photophores of deep sea squid, some with irises and lenses, generate bright multicolored light. Since they also have good vision, there is a potential for high bandwidth communication.

Their behavior is mammal like. Octopus are reclusive and shy, squid are occasionally very aggressive. Small octopus can learn to solve problems like how to open a container of food. Giant squid, with large nervous systems, have hardly ever been observed except as corpses. They might be as clever as whales.

Birds are vertebrates, related to us through a 300 million year old, probably not very bright, early reptile. Size-limited by the dynamics of flying, some are intellectually comparable to the highest mammals.

The intuitive number sense of crows and ravens extends to seven, compared to three or four for us. Birds outperform all mammals except higher primates and the whales in "learning set" tasks, where the idea is to generalize from specific instances. In mammals generalization depends on cerebral cortex size. In birds forebrain regions called the Wulst and the hyperstriatum are critical, while the cortex is small and unimportant.

Our last common ancestor with the whales was a primitive rat-like mammal alive 100 million years ago. Some dolphin species have body and brain masses identical to ours, and have had them for more generations. They are as good as us at many kinds of problem solving, and can grasp and communicate complex ideas. Killer whales have brains five times human size, and their ability to formulate plans is better than the dolphins', who they occasionally eat. Sperm whales, though not the largest animals, have the world's largest brains. Intelligence may be an important part of their struggle with large squid, their main food. Elephant brains are three times human size. Elephants form matriarchal tribal societies and exhibit complex behavior. Indian domestic elephants learn over 500 commands, and form voluntary mutual benefit relationships with their trainers, exchanging labor for baths. They can solve problems such

as how to sneak into a plantation at night to steal bananas, after having been belled (answer: stuff mud into the bells). And they do have long memories.

Apes are our 10 million year cousins. Chimps and gorillas can learn to use tools and to communicate in human sign languages at a retarded level. Chimps have one third, and gorillas one half, human brainsize.

Animals exhibiting near-human behavior have hundred billion neuron nervous systems. Imaging vision alone requires a billion. The smartest insects have a million brain cells, while slugs and worms make do with a thousand, and sessile animals with a hundred. The portions of nervous systems for which tentative wiring diagrams have been obtained, including nearly all of the large neuroned sea slug, Aplysia, the flight controller of the locust and the early stages of vertebrate vision, reveal neurons configured into efficient, clever, assemblies.

Mobility and Intelligence around the Lab

The twenty year old modern robotics effort can hardly hope to rival the billion year history of large life on earth in richness of example or profundity of result. Nevertheless the evolutionary pressures that shaped life are already palpable in the robotics labs. I'm lucky enough to have participated in some of this activity and to have watched more of it at first hand, and so will presume to interpret the experience.

The first serious attempts to link computers to robots involved hand-eye systems, wherein a computer-interfaced camera looked down at a table where a mechanical manipulator operated. The earliest of these (ca. 1965) were built while the small community of artificial intelligence researchers was still flushed with the success of the original AI programs - programs that almost on the first try played games, proved mathematical theorems and solved problems in narrow domains nearly as well as humans. The robot systems were seen as providing a richer medium for these thought processors. Of course, a few minor new problems did come up.

A picture from a camera can be represented in a computer as a rectangular array of numbers, each representing the shade of gray or the color of a point in the image. A good quality picture requires a million such numbers. Identifying people, trees, doors, screwdrivers and teacups in such an undifferentiated mass of numbers is a formidable problem - the first programs did not attempt it. Instead they were restricted to working with bright cubical blocks on a dark tabletop; a caricature of a toddler learning hand-eye co-ordination. In this simplified environment computers more powerful than those that had earlier aced chess, geometry and calculus problems, combined with larger, more developed, programs were able to sometimes, with luck, correctly locate and grab a block.

The general hand-eye systems have now mostly evolved into experiments to study smaller parts of the problem, for example dynamics or force feedback, or into specialized systems aimed at industrial applications. Most arm systems have special grippers, special sensors, and vision systems and controllers that work only in limited domains. Economics favors this, since a fixed arm, say on an

assembly line, repetitively encounters nearly identical conditions. Methods that handle the frequent situations with maximum efficiency beat more expensive general methods that deal with a wide range of circumstances that rarely arise, while performing less well on the common cases.

Shortly after cameras and arms were attached to computers, a few experiments with computer controlled mobile robots were begun. The practical problems of instrumenting and keeping operational a remote controlled, battery powered, camera and video transmitter toting vehicle compounded the already severe practical problems with hand-eye systems, and conspired to keep many potential players out of the game.

The earliest successful result was SRI's Shakey (ca. 1970). Although it existed as a sometimes functional physical robot, Shakey's primary impact was as a thought experiment. Its creators were of the first wave "reasoning machine" branch of AI, and were interested primarily in applying logic based problem solving methods to a real world task. Control and seeing were treated as system functions of the robot and relegated mostly to staff engineers and undergraduates. Shakey physically ran very rarely, and its blocks world based vision system, which reqired that its environment contain only clean walls and a few large smooth prismatic objects, was coded inefficiently and ran very slowly, taking about an hour to find a block and a ramp in a simple scene. Shakey's most impressive performance, physically executed only piecemeal, was to "push the block" in a situation where it found the block on a platform. The sequence of actions included finding a wedge that could serve as a ramp, pushing it against the platform, then driving up the ramp onto the platform to push the block off.

The problems of a mobile robot, even in this constrained an environment inspired and required the development of a powerful, effective, still unmatched, system STRIPS that constructed plans for robot tasks. STRIPS' plans were constructed out of primitive robot actions, each having preconditions for applicability and consequences on completion. It could recover from unexpected glitches by incremental replanning. The unexpected is a major distinguishing feature of the world of a mobile entity, and is one of the evolutionary pressures that channels the mobile towards intelligence.

Mobile robots have other requirements that guide the evolution of their minds away from solutions seemingly suitable for fixed manipulators. Simple visual shape recognition methods are of little use to a machine that travels through a cluttered three dimensional world. Precision mechanical control of position can't be achieved by a vehicle that traverses rough ground. Special grippers don't pay off when many different and unexpected objects must be handled. Linear algorithmic control systems are not adequate for a rover that often encounters surprises in its wanderings.

The Stanford Cart was a mobile robot built about the same time as Shakey, on a lower budget. From the start the emphasis of the Cart project was on low level perception and control rather than planning, and the Cart was actively used as a physical experimental testbed to guide the research. Until its retirement in 1980 it (actually the large mainframe computer that remote controlled it) was programmed to:

- Follow a white line in real time using a TV camera mounted at about eye level on the robot. The program had to find the line in a scene that contained a lot of extraneous imagery, and could afford to digitize only a selected portion of the images it processed.

- Travel down a road in straight lines using points on the horizon as references for its compass heading (the cart carried no instrumentation of any kind other than the TV camera). The program drove it in bursts of one to ten meters, punctuated by 15 second pauses to think about the images and plan the next move.
- Go to desired destinations about 20 meters away (specified as so many meters forward and so many to the left) through messy obstacle courses of arbitrary objects, using the images from the camera to servo the motion and to detect (and avoid) obstacles in three dimensions. With this program the robot moved in meter long steps, thinking about 15 *minutes* before each one. Crossing a large room or a loading dock took about five hours, the lifetime of a charge on the Cart's batteries.

The vision, world representation and planning methods that ultimately worked for the Cart (a number were tried and rejected) were quite different than the "blocks world" and specialized industrial vision methods that grew out of the hand-eye efforts. Blocks world vision was completely inappropriate for the natural indoor and outdoor scenes encountered by the robot. Much experimentation with the Cart eliminated several other initially promising approaches that were insufficiently reliable when fed voluminous and variable data from the robot. The product was a vision system with a different flavor than most. It was "low level" in that it did no object modelling, but by exploiting overlapping redundancies it could map its surroundings in 3D reliably from noisy and uncertain data. The reliability was necessary because Cart journeys consisted of typically twenty moves each a meter long punctuated by vision steps, and each step had to be accurate for the journey to succeed.

At Carnegie-Mellon University we are building on the Cart work with (so far) four different robots optimized for different parts of the research.

Pluto was designed for maximum generality - its wheel system is omnidirectional, allowing motion in any direction while simultaneously permitting the robot to spin like a skater. It was planned that Pluto would continue the line of vision research of the Cart and also support work in close-up navigation with a manipulator (we would like a fully visually guided procedure that permits the robot to find, open and pass through a door). The real world has changed our plans. To our surprise, the problem of controlling the three independently steerable and driveable wheel assemblies of Pluto is an example of a difficult, so far unsolved, problem in control of overconstrained systems. We are working on it, but in the meantime Pluto is nearly immobile.

When the difficulty with Pluto became apparent, we built a simple robot, Neptune, to carry on the long range vision work. I'm happy to announce that Neptune is now able to cross a room in under an hour, five times more quickly than the Cart.

Uranus is the third robot in the CMU line, designed to do well the things that Pluto has so far failed to do. It will achieve omnidirectionality through curious wheels, tired with rollers at 45 degrees, that, mounted like four wagon wheels, can travel forward and backward normally, but that screw themselves sideways when wheels on opposite sides of the robot are turned in opposite directions.

Our fourth mobile robot is called the Terragator, for terrestrial navigator, and is designed to travel outdoors for long distances. It is much bigger than the others, almost as large as a small car, and is powered by a gasoline generator rather than batteries. We expect to program it to travel on roads, avoid and recognize outdoor obstacles and landmarks. Our earlier work makes clear that in order to run at the speeds we have in mind (a few km/hr) we will need processing speeds about 100 times faster than our medium size mainframes now provide. We plan to augment our regular machines with a specialized computer called an array processor to achieve these rates.

Our ambitions for the new robots (go down the hall to the third door, go in, look for a cup and bring it back) has created another pressing need - a computer language in which to concisely specify complex tasks for the rover, and a hardware and software system to embody it. We considered something similar to Stanford's AL arm controlling language from which the commercial languages VAL at Unimation and the more sophisticated AML at IBM were derived.

Paper attempts at defining the structures and primitives required for the mobile application revealed that the linear control structure of these state-of-the-art arm languages was inadequate for a rover. The essential difference is that a rover, in its wanderings, is regularly "surprised" by events it cannot anticipate, but with which it must deal. This requires that contingency routines be activated in arbitrary order, and run concurrently. We are experimenting with a structure where a number of specialist programs communicating via a common data structure called a blackboard are active at the same time, some operating sensors, some controlling effectors, some integrating the results of other modules, and some providing overall direction. As conditions change the priority of the various modules changes, and control may be passed from one to another.

The Psychology of Mobile Robots

Suppose we ask Uranus, equipped with a controller based on the blackboard system mentioned in the last section to, in fact, go down the hall to the third door, go in, look for a cup and bring it back. This will be implemented as a process that looks very much like a program written for the arm control languages (that in turn look very much like Algol, or even Basic), except that the door recognizer routine would probably be activated separately. Consider the following caricature of such a program.

```
MODULE Go-Fetch-Cup
Wake up Door-Recognizer with instructions
     ( On Finding-Door Add 1 to Door-Number
          Record Door-Location )
Record Start-Location
Set Door-Number to 0
While Door-Number < 3 Wall-Follow
Face-Door
IF Door-Open THEN Go-Through-Opening
          ELSE Open-Door-and-Go-Through
Set Cup-Location to result of Look-for-Cup
Travel to Cup-Location
Pickup-Cup at Cup-Location
Travel to Door-Location
```

```
Face-Door
IF Door-Open THEN Go-Through-Opening
        ELSE Open-Door-and-Go-Through
Travel to Start-Location
End
```

So far so good. We activate our program and Uranus obediently begins to trundle down the hall counting doors. It correctly recognizes the first one. The second door, unfortunately is decorated with some garish posters, and the lighting in that part of the corridor is poor, and our experimental door recognizer fails to detect it. The wall follower, however, continues to operate properly and Uranus continues on down the hall, its door count short by one. It recognizes door 3, the one we had asked it to go through, but thinks it is only the second, so continues. The next door is recognized correctly, and is open. The program, thinking it is the third one, faces it and proceeds to go through. This fourth door, sadly, leads to the stairwell, and poor Uranus, unequipped to travel on stairs, is in mortal danger.

Fortunately there is a process in our concurrent programming system called Detect-Cliff that is always running and that checks ground position data posted on the blackboard by the vision processes and also requests sonar and infrared proximity checks on the ground. It combines these, perhaps with an a-priori expectation of finding a cliff set high when operating in dangerous areas, to produce a number that indicates the likelyhood there is a drop-off in the neighborhood.

A companion process Deal-with-Cliff also running continuously, but with low priority, regularly checks this number, and adjusts its own priority on the basis of it. When the cliff probability variable becomes high enough the priority of Deal-with-Cliff will exceed the priority of the current process in control, Go-Fetch-Cup in our example, and Deal-with-Cliff takes over control of the robot. A properly written Deal-with-Cliff will then proceed to stop or greatly slow down the movement of Uranus, to increase the frequency of sensor measurements of the cliff, and to slowly back away from it when it has been reliably identified and located.

Now there's a curious thing about this sequence of actions. A person seeing them, not knowing about the internal mechanisms of the robot might offer the interpretation "First the robot was determined to go through the door, but then it noticed the stairs and became so frightened and preoccupied it forgot all about what it had been doing". Knowing what we do about what really happened in the robot we might be tempted to berate this poor person for using such sloppy anthropomorphic concepts as determinination, fear, preoccupation and forgetfulness in describing the actions of a machine. We could berate the person, but it would be wrong.

I think the robot came by the emotions and foibles indicated as honestly as any living animal. An octopus in pursuit of a meal can be diverted by hints of danger in just the way Uranus was. An octopus also happens to have a nervous system that evolved entirely independently of our own vertebrate version. Yet most of us feel no qualms about ascribing concepts like passion, pleasure, fear and pain to the actions of

the animal.

We have in the behavior of the vertebrate, the mollusc and the robot a case of convergent evolution. The needs of the mobile way of life have conspired in all three instances to create an entity that has modes of operation for different circumstances, and that changes quickly from mode to mode on the basis of uncertain and noisy data prone to misinterpretation. As the complexity of the mobile robots increases I expect their similarity to animals and humans will become even greater.

Among the natural traits I see in the immediate roving robot horizon is parameter adjustment learning. A precision mechanical arm in a rigid environment can usually have its kinematic self-model and its dynamic control parameters adjusted once permanently. A mobile robot bouncing around in the muddy world is likely to continuously suffer insults like dirt buildup, tire wear, frame bends and small mounting bracket slips that mess up accurate a-priori models. Our present visual obstacle course software, for instance, has a camera calibration phase where the robot is parked precisely in front of an exact grid of spots so that a program can determine a function that corrects for distortions in the camera optics. This allows other programs to make precise visual angle measurements in spite of distortions in the cameras. We have noticed that our present code is very sensitive to mis-calibrations, and are working on a method that will continuously calibrate the cameras just from the images perceived on normal trips through clutter. With such a procedure in place, a bump that slightly shifts one of the robot's cameras will no longer cause systematic errors in its navigation. Animals seem to tune most of their nervous systems with processes of this kind, and such accomodation may be a precursor to more general kinds of learning.

Perhaps more controversially, I see the begininnings of self awareness in the robots. All of our control programs have internal representations, at varying levels of abstraction and precision, of the world around the robot, and of the robot's position within that world. The motion planners work with these world models in considering alternative future actions for the robot. If our programs had verbal interfaces we could ask questions that receive answers such as "I turned right because I didn't think I could fit through the opening on the left ". As it is we get the same information in the form of pictures drawn by the programs.

So What's Missing?

There may seem to be a contradiction in the various figures on the speed of computers. Once billed as "Giant Brains" computers can do some things, like arithmetic, millions of times faster than human beings. "Expert systems" doing qualitative reasoning in narrow problem solving areas run on these computers approximately at human speed. Yet it took such a computer five hours to simply drive the Cart across a room, down to an hour for Neptune. How can such numbers be reconciled?

The human evolutionary record provides the clue. While our sensory and muscle control systems have been in development for a billion years, and common sense reasoning has been honed for probably about a million, really high level, deep, thinking is little more than a parlor trick, culturally developed over a few thousand years, which a few humans, operating largely against their natures, can learn. As with

Samuel Johnson's dancing dog, what is amazing is not how well it is done, but that it is done at all.

Computers can challenge humans in intellectual areas, where humans perform inefficiently, because they can be programmed to carry on much less wastefully. An extreme example is arithmetic, a function learned by humans with great difficulty, which is instinctive to computers. These days an average computer can add a million large numbers in a second, which is more than a million times faster than a person, and with no errors. Yet one hundred millionth of the neurons in a human brain, if reorganized into an adder using switching logic design principles, could sum a thousand numbers per second. If the whole brain were organized this way it could do sums one hundred thousand times faster than the computer.

Computers do not challenge humans in perceptual and control areas because these billion year old functions are carried out by large fractions of the nervous system operating as efficiently as the hypothetical neuron adder above. Present day computers, however efficiently programmed, are simply too puny to keep up. Evidence comes from the most extensive piece of reverse engineering yet done on the vertebrate brain, the functional decoding of some of the visual system by D. H. Hubel, T. N. Weisel and colleagues.

The vertebrate retina's 20 million neurons take signals from a million light sensors and combine them in a series of simple operations to detect things like edges, curvature and motion. Then image thus processed goes on to the much bigger visual cortex in the brain.

Assuming the visual cortex does as much computing for its size as the retina, we can estimate the total capability of the system. The optic nerve has a million signal carrying fibers and the optical cortex is a thousand times deeper than the neurons which do a basic retinal operation. The eye can process ten images a second, so the cortex handles the equivalent of 10,000 simple retinal operations a second, or 3 million an hour.

An efficient program running on a typical computer can do the equivalent work of a retinal operation in about two minutes, for a rate of 30 per hour. Thus seeing programs on present day computers seem to be 100,000 times slower than vertebrate vision. The whole brain is about ten times larger than the visual system, so it should be possible to write real-time human equivalent programs for a machine one million times more powerful than todays medium sized computer. Even todays largest supercomputers are about 1000 times slower than this desiratum. How long before our research medium is rich enough for full intelligence?

Since the 1950s computers have gained a factor of 1000 in speed per constant dollar every decade. There are enough developments in the technological pipeline, and certainly enough will, to continue this pace for the forseeable future.

The processing power available to AI programs has not increased proportionately. Hardware speedups and budget increases have been dissipated on convenience features; operating systems, time sharing,

high level languages, compilers, graphics, editors, mail systems, networking, personal machines, etc. and have been spread more thinly over ever greater numbers of users. I believe this hiatus in the growth of processing power explains the disappointing pace of AI in the past 15 years, but nevertheless represents a good investment. Now that basic computing facilities are widely available, and thanks largely to the initiative of the instigators of the Japanese Supercomputer and Fifth Generation Computer projects, attention worldwide is focusing on the problem of processing power for AI.

The new interest in crunch power should insure that AI programs share in the thousandfold per decade increase from now on. This puts the time for human equivalence at twenty years. The smallest vertebrates, shrews and hummingbirds, derive interesting behavior from nervous systems one ten thousandth the size of a human's, so we can expect fair motor and perceptual competence in less than a decade. By my calculations and impressions present robot programs are similar in power to the control systems of insects.

Some principals in the Fifth Generation Project have been quoted as planning "man capable" systems in ten years. I believe this more optimistic projection is unlikely, but not impossible. The fastest present and nascent computers, notably the Cray X-MP and the Cray 2, compute at 109 operations/second, only 1000 times too slowly.

As the computers become more powerful and as research in this area becomes more widespread the rate of visible progress should accelerate. I think artificial intelligence via the "bottom up" approach of technological recapitulation of the evolution of mobile animals is the surest bet because the existence of independently evolved intelligent nervous systems indicates that there is an incremental route to intelligence. It is also possible, of course, that the more traditional "top down" approach will achieve its goals, growing from the narrow problem solvers of today into the much harder areas of learning, common-sense reasoning and perceptual acquisition of knowledge as computers become large and powerful enough, and the techniques are mastered. Most likely both approaches will make enough progress that they can effectively meet somewhere in the middle, for a grand synthesis into a true artificial sentience.

This artificial person will have some interesting properties. Its high level reasoning abilities should be astonishingly better than a human's - even today's puny systems are much better in some areas - but its low level perceptual and motor abilities will be comparable to ours. Most interestingly it will be highly changeable, both on an individual basis and from one of its generations to the next. And it will quickly become cheap.

The Future

What happens when increasingly cheap machines can replace humans in any situation? What will I do when a computer can write this article, and do research, better than me? These questions face some occupations now. They will affect everybody in a few decades.

By design, machines are our obedient and able slaves. But intelligent machines, however benevolent, threaten our existence because they are alternative inhabitants of our ecological niche. Machines merely as clever as human beings will have enormous advantages in competitive situations. Their production and upkeep costs less, so more of them can be put to work with given resources. They can be optimized for their jobs, and programmed to work tirelessly.

Intelligent robots will have even greater advantages away from our usual haunts. Very little of the known universe is suitable for unaided humans. Only by massive machinery can we survive in outer space, on the surfaces of the planets or on the sea floor. Smaller, intelligent but unpeopled, devices will be able to do what needs to be done there more cheaply. The Apollo project put people on the moon for forty billion dollars. Viking landed machines on Mars for one billion. If the Viking landers had been as capable as humans, their multi-year stay would have told us much more about Mars than we found out about the moon from Apollo.

As if this weren't bad enough, the very pace of technology presents an even more serious challenge. We evolved with a leisurely 100 million years between significant changes. The machines are making similar strides in decades. The rate will quicken further as multitudes of cheap machines are put to work as programmers and engineers, with the task of optimizing the software and hardware which makes them what they are. The successive generations of machines produced this way will be increasingly smarter and cheaper. There is no reason to believe that human equivalence represents any sort of upper bound. When pocket calculators can out-think humans, what will a big computer be like? We will simply be outclassed.

Then why rush headlong into the intelligent machine era? Wouldn't any sane human try to delay things as long as possible? The answer is obvious, if unpalatable on the surface. Societies and economies are as surely subject to evolutionary pressures as biological organisms. Failing social systems wither and die, to be replaced by more successful competitors. Those that can sustain the most rapid expansion dominate sooner or later.

We compete with each other for the resources of the accessible universe. If automation is more efficient than hand labor, organizations and societies which embrace it will be wealthier and better able to survive in difficult times, and expand in favorable ones. If the U.S. were to unilaterally halt technological development, an occasionally fashionable idea, it would soon succumb either to the military might of the Soviets, or the economic success of its trading partners. Either way the social ideals that led to the decision would become unimportant on a world scale.

If, by some evil and unlikely miracle, the whole human race decided to eschew progress, the long term result would be almost certain extinction. The universe is one random event after another. Sooner or later an unstoppable virus deadly to humans will evolve, or a major asteroid will collide with the earth, or the sun will go nova, or we will be invaded from the stars, or a black hole will swallow the galaxy.

The bigger, more diverse and competent a culture is, the better it can detect and deal with external dangers. The bigger events happen less frequently. By growing sufficiently rapidly it has a finite chance of surviving forever. Even the eventual collapse or heat death of the universe might be evaded or survived if an entity can restructure itself properly.

The human race will expand into the solar system soon, and human occupied space colonies will be part of it. But the economics of automation will become very persuasive in space even before machines achieve human competence.

I visualize immensely lucrative self-reproducing robot factories in the asteroids. Solar powered machines would prospect and deliver raw materials to huge, unenclosed, automatic processing plants. Metals, semiconductors and plastics produced there would be converted by robots into components which would be assembled into other robots and structural parts for more plants. Machines would be recycled as they broke. If the reproduction rate is higher than the wear out rate, the system will grow exponentially. A small fraction of the output of materials, components, and whole robots could make someone very, very rich.

The first space industries will be more conventional. Raw materials purchased from Earth or from human space settlements will be processed by human supervised machines and sold at a profit. The high cost of maintaining humans in space insures that that there will always be more machinery per person there than on Earth. As machines become more capable, the economics will favor an ever higher machine/people ratio. Humans will not necessarily become fewer, but the machines will multiply faster.

When humans become unnecessary in space industry, the machines' physical growth rate will climb. When machines reach and surpass humans in intelligence, the intellectual growth rate will rise similarly. The scientific and technical discoveries of super-intelligent mechanisms will be applied to making themselves smarter still. The machines, looking quite unlike the machines we know, will explode into the universe, leaving us behind in a figurative cloud of dust. Our intellectual, but not genetic, progeny will inherit the universe. Barring prior claims.

This may not be as bad as it sounds, since the machine civilization will certainly take along everything we consider important, including the information in our minds and genes. Real live human beings, and a whole human community, could be reconstituted if an appropriate circumstance ever arose. Since we are biologically committed to personal death, immortal only through our children and our culture, shouldn't we rejoice to see that culture become as robust as possible?

An Alternative

Some of us have very egocentric world views. We anticipate the discovery, within our lifetimes, of methods to extend human lifespans, and look forward to a few eons of exploring the universe. We don't take kindly to being upstaged by our creations.

The machines' major advantage is their progress rate. Our evolution is largely cultural, but is tightly constrained by our Darwinianly evolving biological substrate. Machinery evolves 100% culturally, culture itself being a rapidly evolving process that feeds on and accelerates itself. How can we, personally, become full, unhandicapped, players in this new game?

Genetic engineering is an option. Successive generations of human beings could be designed by mathematics, computer simulations, and experimentation, like airplanes and computers are now. But this is just building robots out of protein. Away from Earth, protein is not an ideal material. It's stable only in a narrow temperature and pressure range, is sensitive to high energy disturbances, and rules out many construction techniques and components. Anyway, second rate superhuman beings are just as threatening as first rate ones, of whatever they're made.

What's really needed is a process that gives an individual all the advantages of the machines, at small personal cost. Transplantation of human brains into manufactured bodies has some merit, because the body can be matched to the environment. It does nothing about the limited and fixed intelligence of the brain, which the artificial intellects will surpass.

Transmigration

You are in an operating room. A robot brain surgeon is in attendance. By your side is a potentially human equivalent computer, dormant for lack of a program to run. Your skull, but not your brain, is anesthetized. You are fully conscious. The surgeon opens your brain case and peers inside. Its attention is directed at a small clump of about 100 neurons somewhere near the surface. It determines the three dimensional structure and chemical makeup of that clump non-destructively with high resolution 3D NMR holography, phased array radio encephalography, and ultrasonic radar. It writes a program that models the behavior of the clump, and starts it running on a small portion of the computer next to you. Fine connections are run from the edges of the neuron assembly to the computer, providing the simulation with the same inputs as the neurons. You and the surgeon check the accuracy of the simulation. After you are satisfied, tiny relays are inserted between the edges of the clump and the rest of the brain. Initially these leave brain unchanged, but on command they can connect the simulation in place of the clump. A button which activates the relays when pressed is placed in your hand. You press it, release it and press it again. There should be no difference. As soon as you are satisfied, the simulation connection is established firmly, and the now unconnected clump of neurons is removed. The process is repeated over and over for adjoining clumps, until the entire brain has been dealt with. Occasionally several clump simulations are combined into a single equivalent but more efficient program. Though you have not lost consciousness, or even your train of thought, your mind (some would say soul) has been removed from the brain and transferred to a machine.

In a final step your old body is disconnected. The computer is installed in a shiny new one, in the style, color and material of your choice. You are no longer a cyborg halfbreed, your metamorphosis is complete.

For the squeamish there are other ways to work the transfer. The high resolution brain scan could be done in one fell swoop, without surgery, and a new you made, "While-U-Wait". Some will object that the instant process makes only a copy, the real you is still trapped in the old body (please dispose of properly). This is an understandable misconception growing from the intimate association of a person's identity with a particular, unique, irreplaceable piece of meat. Once the possibility of mind transfer is accepted, however, a more mature notion of life and identity becomes possible. You are not dead until the last copy is erased; a faithful copy is exactly as good as the original.

If even the last technique is too invasive for you, imagine a more psychological approach. A kind of pocket computer (perhaps shaped and worn like glasses) is programmed with the universals of human mentality, with your genetic makeup and with whatever details of your life are conveniently available. It carries a program that makes it an excellent mimic. You carry this computer with you through the prime of your life, and it diligently listens and watches, and perhaps monitors your brainwaves, and learns to anticipate your every move and response. Soon it is able to fool your friends on the phone with its convincing imitation of you. When you die it is installed in a mechanical body and smoothly and seamlessly takes over your life and responsibilities.

What? Still not satisfied? If you happen to be a vertebrate there is another option that combines some of the sales features of the methods above. The vertebrate brain is split into two hemispheres connected by a very large bundle of nerve fibers called the corpus callosum. When brain surgery was new it was discovered that severing this connection between the brain halves cured some forms of epilepsy. An amazing aspect of the procedure was the apparent lack of side effects on the patient. The corpus callosum is a bundle far thicker than the optic nerve or even the spinal cord. Cut the optic nerve and the victim is utterly blind; sever the spinal cord and the body goes limp. Slice the huge cable between the hemispheres and nobody notices a thing. Well, not quite. In subtle experiments it was noted that patients who had this surgery were unable, when presented with the written word "brush", for instance, to identify the object in a collection of others using their left hand. The hand wanders uncertainly from object to object, seemingly unable to decide which is "brush". When asked to do the same task with the right hand, the choice is quick and unhesitating. Sometimes in the left handed version of the task the right hand, apparently in exasperation, reaches over to guide the left to the proper location. Other such quirks involving spatial reasoning and motor co-ordination were observed.

The explanation offered is that the callosum indeed is the main communications channel between the brain hemispheres. It has fibers running to every part of the cortex on each side. The brain halves, however, are fully able to function separately, and call on this channel only when a task involving co-ordination becomes necessary. We can postulate that each hemisphere has its own priorities, and that the other can request, but not demand, information or action from it, and must be able to operate effectively if the other chooses not to respond, even when the callosum is intact. The left hemisphere handles language and controls the right side of the body. The right hemisphere controls the left half of the body, and without the callosum the correct interpretation of the letters "b r u s h" could not be conveyed to the controller of the left hand.

But what an opportunity. Suppose we sever your callosum but then connect a cable to both severed ends leading into an external computer. If the human brain is understood well enough this external computer can be programmed to pass, but also monitor the traffic between the two. Like the personal mimic it can teach itself to think like them. After a while it can insert its own messages into the stream, becoming an integral part of your thought processes. In time, as your original brain fades away from natural causes, it can smoothly take over the lost functions, and ultimately your mind finds itself in the computer. With advances in high resolution scanning it may even be possible to have this effect without messy surgery - you would just wear some kind of helmet or headband.

Whatever style you choose, when the process is complete advantages become apparent. Your computer has a control labelled speed. It had been set to slow, to keep the simulations synchronized with the old brain, but now you change it to fast. You can communicate, react and think a thousand times faster. But wait, there's more!

The program in your machine can be read out and altered, letting you conveniently examine, modify, improve and extend yourself. The entire program may be copied into similar machines, giving two or more thinking, feeling versions of you. You may choose to move your mind from one computer to another more technically advanced, or more suited to a new environment. The program can also be copied to some future equivalent of magnetic tape. If the machine you inhabit is fatally clobbered, the tape can be read into an blank computer, resulting in another you, minus the experiences since the copy. With enough copies, permanent death would be very unlikely.

As a computer program, your mind can travel over information channels. A laser can send it from one computer to another across great distances and other barriers. If you found life on a neutron star, and wished to make a field trip, you might devise a way to build a neutron computer and robot body on the surface, then transmit your mind to it. Nuclear reactions are a million times quicker than chemistry, so the neutron you can probably think that much faster. It can act, acquire new experiences and memories, then beam its mind back home. The original body could be kept dormant during the trip to be reactivated with the new memories when the return message arrived. Alternatively, the original might remain active. There would then be two separate versions of you, with different memories for the trip interval.

Two sets of memories can be merged, if mind programs are adequately understood. To prevent confusion, memories of events would indicate in which body they happened. Merging should be possible not only between two versions of the same individual but also between different persons. Selective mergings, involving some of the other person's memories, and not others would be a very superior form of communication, in which recollections, skills, attitudes and personalities can be rapidly and effectively shared.

Your new body will be able to carry more memories than your original biological one, but the accelerated information explosion will insure the impossibility of lugging around all of civilization's knowledge. You will have to pick and choose what your mind contains at any one time. There will often be knowledge and

skills available from others superior to your own, and the incentive to substitute those talents for yours will be overwhelming. In the long run you will remember mostly other people's experiences, while memories you originated will be floating around the population at large. The very concept of you will become fuzzy, replaced by larger, communal egos.

Mind transferral need not be limited to human beings. Earth has other species with brains as large, from dolphins, our cephalic equals, to elephants, whales, and giant squid, with brains up to twenty times as big. Translation between their mental representation and ours is a technical problem comparable to converting our minds into a computer program. Our culture could be fused with theirs, we could incorporate each other's memories, and the species boundaries would fade. Non-intelligent creatures could also be popped into the data banks. The simplest organisms might contribute little more than the information in their DNA. In this way our future selves will benefit from all the lessons learned by terrestrial biological and cultural evolution. This is a far more secure form of storage than the present one, where genes and ideas are lost when the conditions that gave rise to them change.

Our speculation ends in a super-civilization, the synthesis of all solar system life, constantly improving and extending itself, spreading outwards from the sun, converting non-life into mind. There may be other such bubbles expanding from elsewhere. What happens when we meet? Fusion of us with them is a possibility, requiring only a translation scheme between the memory representations. This process, possibly occuring now elsewhere, might convert the entire universe into an extended thinking entity, a probable prelude to greater things.

Publications

Publications of the CMU Mobile Robot Lab

January 8, 1986

- Elfes, A. E, ***A Sonar-Based Mapping and Navigation System***, Workshop on Robotics, Oak Ridge National Lab, Oak Ridge, TN, August, 1985. (invited presentation), In the proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, April 7-10 1986. (to appear)
- Wallace, R. W., K. Matsuzaki, Y. Goto, J. Webb, J. Crisman, T. Kanade, ***Progress in Robot Road Following***, the proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, April 7-10 1986. (to appear)
- The CMU Mobile Robot Lab staff, ***Towards Autonomous Vehicles***, CMU Robotics Institute 1984 Annual Research Review, The Robotics Institute, CMU, Pittsburgh, PA, September 1985, pp. 33-50.
- Moravec, H. P., ***The Second Week: Speculations on the future of artificial and human intelligence***, Harvard University Press, Cambridge, 1986. (in preparation)
- Muir, P. F. and C. P. Neuman, ***Pulse-Width Modulation Control of Brushless DC Motors for Robotic Applications***, Presented at the 27th Midwest Symposium on Circuits and Systems June 12, 1984, Morgantown, West Virginia. In IEEE Transactions on Industrial Electronics, August 1985, pp. 222-229.
- Muir, P. F. and C. P. Newman, ***Kinematic Modelling of Wheeled Mobile Robots***, 1985. (in preparation)
- Thorpe, C., L. Matthies and H. Moravec, ***Experiments and Thoughts on Visual Navigation***, the proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, March, 1985, pp. 830-835.
- Wallace, R. S., ***A Modified Hough Transform for Lines***, proceedings of the 1985 IEEE Conference on Vision and Pattern Recognition, San Francisco, June, 1985, pp. 665-667.
- Wallace, R., A. Stentz, C. Thorpe, H. Moravec, W. Whittaker and T. Kanade, ***First Results in Robot Road-Following***, proceedings of the 1985 IJCAI, Los Angeles, August, 1985 and proceedings of the 1985 ASME conference on Computers in Engineering, Boston, August, 1985.
- Moravec, H. P. and L. H. Matthies, eds., ***Vision, Planning and Control for Mobile Robots***, Kluwer Academic Publishers, 1986. (in preparation)
- Moravec, H. P. and A. E. Elfes, ***High Resolution Maps from Wide Angle Sonar***, proceeding of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, March, 1985, pp 116-121, and proceedings of the 1985 ASME conference on Computers in Engineering, Boston, August, 1985.
- Moravec, H. P., ***Machines with Mobility***, In ***Robots***, Salamander Books, London, England 1985, pp. 66-79. American edition by Crescent Books.
- Moravec, H. P., ***The Rovers***, In ***Robotics***, Marvin Minsky, ed., Doubleday, 1985, pp. 123-145.
- Podnar, G. W., K. F. Hensley and M. K. Blackwell, ***Physical System of a Mobile Robot: Pluto***, CMU Robotics Institute technical report, 1985. (in preparation)

- Thorpe, C. E., ***FIDO: Vision and Navigation for a Mobile Robot***, PhD Thesis, Computer Science Dept., Carnegie-Mellon University, Autumn 1984.
- Wallace, R. S., ***Three Findpath Problems***, Proceeding of AAAI-84, Austin, Texas, August 6-10, 1984, pp 326-329.
- Lucas, B. D., ***Generalized Image Matching by the Method of Differences***, Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon University, Autumn 1984.
- Moravec, H. P., ***Locomotion, Vision and Intelligence***, First International Symposium of Robotics Research, Bretton Woods, N.H., August 1983. (invited presentation) in ***Robotics Research - The First International Symposium***, Michael Brady and Richard Paul, eds., MIT Press, 1984, pp. 215-224.
- Moravec, H. P., ***Three Degrees for a Mobile Robot***, Proceedings of the ASME Conference on Computers in Engineering, Las Vegas, Nevada, August, 1984.
- Matthies, L. H. and C. E. Thorpe, ***Experience with Visual Robot Navigation***, proceedings of IEEE Oceans 84, Washington, D.C., August, 1984.
- Thorpe, C. E., ***Path Relaxation: Path Planning for a Mobile Robot***, CMU-RI-TR-84-5, Robotics Institute, Carnegie-Mellon University, April, 1984. also in proceedings of IEEE Oceans 84, Washington, D.C., August, 1984 and Proceedings of AAAI-84, Austin, Texas, August 6-10, 1984, pp. 318-321.
- Podnar, G. W., K. Dowling and M. K. Blackwell, ***A Functional Vehicle for Autonomous Mobile Robot Research***, CMU Robotics Institute technical report CMU-RI-TR-84-28, April 1984.
- Muir, P. F., ***Digital Servo Controller Design for Brushless D.C. Motors***, Master's Thesis, Carnegie-Mellon University, April, 1984. July 1984, CMU-Robotics Technical report.
- Thorpe, C. E., ***An Analysis of Interest Operators for FIDO***, CMU-RI-TR-83-19, Robotics Institute, Carnegie-Mellon University, December 14, 1983 also in Proceeding of IEEE Workshop on Computer Vision, 1984.
- Moravec, H. P., ***Mobile Robots: Basic Research***, Workshop on Autonomous Ground Vehicles, Xerox International Center, Leesburg, VA, October 24-26, 1983. (invited presentation)
- Elfes, A. and S. N. Talukdar, ***A Distributed Control System for the CMU Rover***, proceedings of the eighth IJCAI-83, Karlsruhe, West Germany, August 8-12, 1983, pp. 830-833.
- Thorpe, C. E. and S. A. Shafer, ***Correspondence in Line Drawings of Multiple Views of Objects***, Proceedings of IJCAI-83, Karlsruhe, West Germany, August 8-12, 1983, pp. 959-965.
- Elfes, A. and S. N. Talukdar, ***A Distributed Control System for a Mobile Robot***, proceedings of the first National Congress on Industrial Automation, Sao Paulo, Brasil, July 11-15, 1983, pp 360-366 and CMU Design Research Center report DRC-18-65-83, December 1983.
- Moravec, H. P., ***The Stanford Cart and The CMU Rover***, Proceedings of the IEEE, July 1983, pp. 872-884. (invited paper)
- Thorpe, C. E., ***The CMU Rover and the FIDO Vision and Navigation System***, 1983 Symposium on Autonomous Underwater Robots, University of New Hampshire, Marine Systems Engineering Lab, May, 1983.
- Carley, F. B. and H. P. Moravec, ***The Rocket/Skyhook Combination***, *L5 News*, Vol. 8 #3, March 1983, pp. 4-6.

- Thorpe, C. E. and D. M. McKeown, ***Autonomous Vehicle Navigation: Smart Vehicle Simulator Version 1*** Technical Report, Westinghouse Electric Corporation, December 1982.
- Thorpe, C. E., ***Underwater Landmark Identification***, Proceedings of the Second International Computer Engineering Conference American Society of Mechanical Engineers, August 1982.
- Moravec, H. P., ***The CMU Rover*** Proceedings of AAAI-82, the second national artificial intelligence conference, Pittsburgh, August 18-20, 1982, pp. 377-380. also *Update*, Volume 3, July 1982, Marine Systems Engineering Laboratory, University of New Hampshire.
- Moravec, H. P., ***The Endless Frontier and The Thinking Machine***, in **The Endless Frontier**, Vol. 2, Jerry Pournelle, ed., Grosset & Dunlap, Ace books, January 1982, pp. 374-397.
- Lucas, B. D. and T. Kanade, ***An Iterative Image Registration Technique with an Application to Stereo Vision***, proceedings of the seventh International Joint Conference on Artificial Intelligence, Vancouver, British Columbia, August 1981.
- Moravec, H. P., ***Rover Visual Obstacle Avoidance***, proceedings of the seventh International Joint Conference on Artificial Intelligence, Vancouver, British Columbia, August 1981, pp. 785-790.
- Moravec, H. P., ***3D Graphics and the Wave Theory***, presented at the 1981 Siggraph conference, Dallas, Texas, August 1981; *Computer Graphics*, Vol. 15 #3, August 1981, pp. 289-296.
- Moravec, H. P., ***Robot Rover Visual Navigation***, UMI Research Press, Ann Arbor, Michigan, 1981. (book)
- Forward, R. L. and H. P. Moravec, ***High Wire Act***, *Omni*, Omni publications international, New York, July 1981, pp. 44-47. Also in ***The OMNI Book of Space***, Zebra Books, Kensington Publishing Corp., New York, N. Y., January, 1984, pp. 73-82.
- Moravec, H. P., ***Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover***, Ph.D. thesis, Stanford University, Stanford, California, May 1980. Available as Stanford AIM-340, CS-80-813 and CMU-RI-TR-3.

Distribution List

Defense Documentation Center (12 copies)
Cameron Station
Alexandria, VA 22314

Office of Naval Research (2 copies)
Information Systems Program (437)
Arlington, VA 22217

Office of Naval Research
Code 200
Arlington, VA 22217

Office of Naval Research
Code 455
Arlington, VA 22217

Office of Naval Research
Code 458
Arlington, VA 22217

Office of Naval Research
Eastern/Central Regional Office
Bldg 114, Section D
666 Summer Street
Boston, MA 02210

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605

Office of Naval Research
Western Regional Office
1030 East Green Street
Pasadena, CA 91106

Naval Research Laboratory (6 copies)
Technical Information Division, Code 2627
Washington, D.C. 20375

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps (Code RD-1)
Washington, D.C. 20380

Naval Ocean Systems Center
Advanced Software Technology Division
Code 5200
San Diego, CA 92152

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, MD 20084

Captain Grace M. Hopper (008)
Naval Data Automation Command
Washington Navy Yard
Building 166
Washington, DC 20374

END

FILMED

3-86

DTIC